VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

SYSTÉM PRO ON-LINE ZÁZNAM JÍZD AUTOŠKOLY

A SYSTEM FOR ON-LINE DRIVING SCHOOL TRIP RECORDING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. JAKUB VONEŠ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. RADEK BURGET, Ph.D.**
SUPERVISOR

BRNO 2017

Zadání diplomové práce/19403/2016/xvones02

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2016/2017

Zadání diplomové práce

Řešitel: **Voneš Jakub, Bc.**
Obor: Informační systémy
Téma: **Systém pro on-line záznam jízd autoškoly**
A System for On-Line Driving School Trip Recording
Kategorie: Informační systémy

Pokyny:

1. Seznamte se s projektem DoAutoškoly.cz pro zvyšování kvality autoškol pomocí sdílení uživatelských zkušeností
2. Prostudujte nástroje a techniky pro tvorbu aplikací pro mobilní zařízení se zaměřením na platformu Android.
3. Navrhněte aplikaci typu klient-server pro sběr GPS dat v reálném čase za účelem záznamu průběhu jízdy na serveru.
4. Po dohodě s vedoucím implementujte klientskou mobilní aplikaci a příslušnou serverovou část pomocí vhodných technologií.
5. Implementujte možnost sledování jízdy v reálném čase, prohlížení uložených jízd a dodatečné ruční i automatické úpravy uložených dat.
6. Proveďte testování aplikace a zhodnoťte dosažené výsledky.

Literatura:

- Allen, G.: Android 4 - Průvodce programováním mobilních aplikací, Computer Press, 2013
- Lacko, L.: Vývoj aplikací pro Android, Computer Press, 2015
- Gutmans, A., Rethans, D., Bakken, S.: Mistrovství v PHP 5, Computer Press, 2012

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Burget Radek, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 24. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Předmětem této diplomové práce je návrh a realizace aplikace typu klient-server pro sběr GPS dat pomocí chytrého telefonu s operačním systémem Android za účelem sledování v reálném čase a záznamu průběhu jízd na serveru, kde jsou tato data následně upravována podle reálné silniční komunikace. Tato práce tematicky navazuje a rozšiřuje projekt *DoAutoškoly.cz*. Práce obsahuje teoretické seznámení s GPS, problematikou komunikace v reálném čase a platformou Android, včetně aplikačního rozhraní pro GPS. V další části práce jsou tyto poznatky využity k návrhu celého systému. Text diplomové práce následně plynule přechází v implementaci mobilní aplikace, webového modulu a příslušné serverové aplikace. Nadále jsou vyčleněny kapitoly zabývající se sledováním v reálném čase a editováním tras. Závěr práce je věnován testování a vyhodnocení dosažených cílů.

Abstract

The subject of this thesis is the design and implementation of a client-server applications for collecting GPS data using a smartphone with Android OS. The system is used for real-time monitoring and recording vehicle's movement on a server, where the data are adapted to the real road. This thesis is a continuation of the project *DoAutoškoly.cz*. The theoretical part of the thesis deals with GPS, real-time communication problems and the Android platform, including its application interface for GPS. In the next part of the thesis, this knowledge is used to design the whole system. The thesis continues with the implementation of a mobile application, the web module and the corresponding server application. The chapters continue to deal with real-time tracking and route editing. The last part is focused on testing and evaluation of the results.

Klíčová slova

Mobilní aplikace, Android OS, GPS, záznam průběhu jízdy, autoškoly, editace tras, klient-server, komunikace v reálném čase, Socket.IO, Node.js

Keywords

Mobile application, Android OS, GPS, vehicle's movement recording, Driving School, route editing, client-server, real-time communication, Socket.IO, Node.js

Citace

VONEŠ, Jakub. *Systém pro on-line záznam jízd autoškoly*. Brno, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Burget Radek.

Systém pro on-line záznam jízd autoškoly

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Radka Burgeta, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jakub Voneš
17. května 2017

Poděkování

Tímto bych rád poděkoval svému vedoucímu práce Ing. Radku Burgetovi, Ph.D. za jeho odborné vedení a rady, vstřícné jednání, trpělivost, ochotu a čas, který mi věnoval při řešení dané problematiky během konzultací. V neposlední řadě bych rád poděkoval svým rodičům a přítelkyni za podporu a pomoc během celé doby mého studia.

Obsah

Kapitola 1

Úvod

Žijeme ve světě, kde vládnou sociální sítě. Ve světě internetového šílenství často dáváme na rady anonymních lidí, kteří mohou být vzdáleni na míle daleko. Pokud v recenzi konkrétní služby nebo produktu nenacházíme minimálně pět hvězdiček z pěti jakoby pro nás ani neexistovala, a je jedno, že nabídka je daleko příznivější.

Prakticky každý jedinec má v dnešní uspěchané době chytré zařízení, které nosí neustále u sebe. Dokonce můžeme konstatovat, že vlastnictví takového přístroje je dnes téměř samozřejmostí. Drtivá většina těchto zařízení disponuje možností přístupu k internetu nebo zisku informace o zeměpisné poloze přístroje. Tato skutečnost pobízí ke globálnímu shromažďování dat, k jejich analýze a následné prezentaci. Díky těmto informacím můžou poskytovatelé služeb nabídnout svým potenciálním zákazníkům dokonalé prostředky k tomu, aby rychle a efektivně vyhledali vše, co právě potřebují. V dnešní době jsou moderní srovnávací systémy služeb a produktů, které kategorizují nabídku z několika pohledů. Nemusí se ani jednat o konkrétní slovní hodnocení nebo ceny služeb, ale klasifikace lze vytvořit i z naměřených dat, které se následně analyzují a prezentují.

S tímto tématem úzce souvisí i současná situace autoškol v ČR. Téma, které zná téměř každý. Pokud máme kurz o řidičské oprávnění úspěšně za sebou, předáváme naše zkušenosti dále. Budoucí studenti se většinou snaží najít nejlépe a objektivně hodnocenou autoškolu od jednotlivých absolventů. Bohužel trh s nabídkou řidičského oprávnění je u nás tak přehlcený, že vybrat si tu svoji správnou autoškolu je velmi obtížné a časově náročné.

Řešením této problematiky se zabývá Ing. Jan Hrivnák v rámci své diplomové práce na téma *„Zvyšování kvality autoškol pomocí sdílení uživatelských zkušeností“*. [17] Výstupem jeho díla bylo vytvoření webového portálu v oblasti autoškol působících v České republice, jehož úkolem je zvyšovat nejen jejich kvalitu jako celku, ale také kvalitu instruktorů prostřednictvím proškolených žáků a veřejnosti.

Tato diplomová práce tematicky navazuje a rozšiřuje projekt *DoAutoškoly.cz*. Vytvořený systém na základě zadání umožňuje žákům autoškol zaznamenávat své jízdy pomocí chytrého zařízení s platformou *Android* a integrovaným GPS modulem. Student díky naměřeným trasám nabude vědomí, že udělal maximum pro své závěrečné jízdy. Dále se může ujistit, že v rámci cvičení s instruktorem si projel většinu záludných míst ve městě. Zároveň si pomocí map může projít místa, ve kterých si nebyl jistý.

Primárním cílem těchto měření není pouhé zobrazení tras na mapě, ale zejména následné analýzy jízd. Žáci, kteří si teprve vybírají svoji budoucí autoškolu, budou mít díky statistickým údajům větší přehled a lépe se rozhodnou, komu se svěřit do rukou. Lépe řečeno, kde mají šanci se naučit maximum a kde jim instruktoři budou rádi předávat svoje zkušenosti a nebudou je obírat o čas a peníze.

Kapitola 2

Projekt DoAutoškoly.cz

Praktická část této diplomové práce se zabývá zejména návrhem aplikace typu *klient-server* pro sběr GPS dat v reálném čase za účelem záznamu průběhu jízdy na serveru. Dále se zaměřuje na analýzu možných problémů a na následnou implementaci klientské a serverové části. V této kapitole je nutné čtenáře nejprve seznámit se samotným projektem *DoAutoškoly.cz*[1], včetně jeho realizace, uživatelského rozhraní a také diskutovat problematiku autoškol v České republice.

2.1 Problematika autoškol v ČR

V dnešní době je pohlíženo na jednotlivé studenty autoškoly jako na zákazníky, na které se vůbec neberou ohledy. Z nabízené výuky se stala pouze vidina finančního obnosu.

Základním problémem je, že na trhu se vyskytuje obrovské množství autoškol, jejichž finanční rozpočet je z důvodu konkurence velmi omezený. Proto dochází k tomu, že řada autoškol výuku provozuje velmi nepoctivě a tím pádem produkují i velmi podprůměrné řidiče. Bohužel většina těchto absolventů nabude dojmu, že po udělení řidičského oprávnění jsou velmi zdatní v řízení vozidla, ale opak je pravdou. Díky spojení vlastního přecenění řidičských dovedností a předchozího špatného výcviku, dochází mnohdy i ke smrtelným dopravním nehodám.

Důvodem velkého množství autoškol na trhu spočívá v samotných kontrolách, které probíhají pouze na formální úrovni. Jediná kontrola proběhne formou účetnictví a povolení. Nikdo nekontroluje, zda si daný žák vyzkoušel většinu silničních situacích nebo jeho lekce byla využita jen na hodinové stání v koloně. Také se nikdo nezabývá a neověřuje pedagogické kvality lektorů, nikdo neporovnává množství úspěšných absolventů u jednotlivých autoškol a nezajímá je fakt, že existují autoškoly, u nichž většina jejich studentů neuspěje u závěrečné zkoušky. A už vůbec nikdo nekontroluje, zda instruktoři z finančních důvodů nepřeskočí technické školení či školení první pomoci. Existence těchto autoškol díky nízkým cenám nabízených kurzů není ohrožená a má dostatek žáků pro nepoctivou výuku budoucích řidičů. [17]

Na trhu existují autoškoly, kde působí pouze jeden člověk v pozici majitele a současně lektora, který navíc vlastní pouze jedno vozidlo, se kterým realizuje výuku. Na straně druhé existují autoškoly s vlastním vozovým parkem, s pravidelně proškolenými zaměstnanci a s profesionálními učebnami. Ale nakonec jsme zase na samém začátku této problematiky,

[1] https://www.doautoskoly.cz/

protože na konkurenčním trhu autoškol hraje velkou roli cena za výuku, která je většinou pro žáka rozhodující.

2.2 O projektu DoAutoškoly.cz

Už dávno nejsme v době, kdy kolem nás projelo auto s frekvencí jednou za den. S přibývajícími vozidly na silnicích přibývá také i více nepřehledných situací a ve výsledku i kolizí. Tím spíše stačí, aby se na vozovce objevil pouhý jeden nezkušený řidič a problém je na světě. Každý majitel autoškoly by si měl uvědomit, že vozidlo není pouhý dopravní prostředek, ale svým způsobem se také jedná o pomyslnou zbraň, která může při špatném zacházení třeba i zabíjet. V případě, že instruktor zanedbá či podcení svoji práci, může dojít k ohrožení nejenom řidiče, ale také ostatních účastníků silničního provozu.

Primárním cílem projektu *DoAutoškoly.cz* je zvýšení dovedností začínajících řidičů, kteří teprve svoji cestu začínají v některé z vybraných autoškol. Výběr správné autoškoly je zajisté rozhodující faktor, který do budoucna ovlivní jejich dovednosti a schopnosti. Kvalitní instruktor by nás měl připravit na každodenní nástrahy, které nás čekají nejenom na silnicích, ale také by nás měl naučit základní údržbu našeho dopravního prostředku, technické dovednosti a v neposlední řadě i základy první pomoci pro případ záchrany života při dopravní nehodě. Nepoctivý instruktor se ovšem postará pouze o udělení řidičského oprávnění, ale vůbec ho nezajímá, zda se ze studenta stane opravdu kvalitní a zkušený řidič, který si dokáže poradit i v obtížných situacích. [16]

Webový katalog *DoAutoškoly.cz* se snaží ukázat pomocí přehledného uživatelského rozhraní, slovního a číselného hodnocení jednotlivých autoškol, která škola produkuje úspěšné řidiče a která nikoli. [16]

Webová služba vycházející z diplomové práce Ing. Jana Hrivnáka pomáhá naplňovat tyto cíle: [17]

- zvyšovat povědomí o kvalitních autoškolách a zároveň upozornit na ty nekorektní,
- informovat veřejnost o důležitosti kvality výuky,
- pomoc prosadit se těm autoškolám, které dodržují všechny aspekty výuky a nesnižují cenu na úkor technické přípravy a první pomoci,
- pomoc zviditelnit autoškoly, které se stále snaží vychovávat kvalitní a ohleduplné řidiče.

2.3 Struktura webového rozhraní

Při návrhu systému pro on-line záznam jízd je nutné brát také ohled na to, že do současného webového portálu bude nutné přidat možnost sledování on-line jízd, procházení naměřených tras a také jejich editace. Z tohoto důvodu se dále zaměřím na jednotlivé části webu z pohledu uživatelů.

Webová služba *DoAutoškoly.cz* je z pohledu uživatele rozdělena na 4 části, všichni ale přistupují na web pod stejnou doménou druhého řádu. Konkrétně se jedná o tyto části: [17]

1. **Katalog** – Umožňuje vyhledávat autoškoly podle zadaných kritérií, prohlížet jejich detail, dále je mezi sebou porovnávat a také přidávat nová hodnocení. Tato část je tedy zaměřená na uživatele, kteří hledají vhodnou autoškolu ve svém městě. Veškeré

filtrovaní naleznou na úvodní stránce webu. Tito návštěvníci mohou kontaktovat vybrané školy a následně se k ní také přihlásit.

2. **Student** – Je část webu, kde si uživatel vybere konkrétní autoškolu a začne provádět svůj výcvik. Tento žák konkrétně navazuje na předchozí typ uživatele, ale s tím rozdílem, že návštěvník má jistý potenciál se průběžně vracet na webové stránky pod svým unikátním jménem v době svého kurzu. Ve výsledku jsou uživatelské části Katalog a Student sloučeny.

3. **Správa autoškoly** – Tato část webu slouží pouze pro majitele a instruktory jednotlivých autoškol, kteří zde mohou spravovat informace o své autoškole.

4. **Administrace** – Je oddělená část portálu, která slouží výhradně administrátorovi webu pro správu jednotlivých autoškol a kontrolu dat zadaných od předchozích uživatelů webu.

2.4 Implementace a použité technologie

V závislosti na zadání je nutné rozšířit i samotnou webovou aplikaci, kde příslušní uživatelé budou mít k dispozici například upravování naměřených tras nebo on-line sledování jízd. Z tohoto důvodu v této podkapitole rozeberu použité technologie a způsob implementace webového portálu *DoAutoškoly.cz*.

Front-end

Pro samotné zobrazení webu u klienta byla použita nejnovější verze značkovacího jazyku *HTML*, tedy *HTML 5*. Vzhledem k tomu, že se jedná o jazyk pro statickou tvorbu webových stránek, byl na klientské straně přidán skriptovací jazyk *JavaScript*, konkrétně ve verzi *ECMAScript*. Pro usnadnění práce s JavaScriptem je využita jedna z nejpoužívanějších knihoven *jQuery*. Vzhledem k tomu, že se jedná vcelku o jednoduché uživatelské rozhraní, nebyl použit žádný JavaScriptový framework pro urychlení práce, jako je například *AngularJS*[2] nebo *React.JS*[3]. Dokonce nebyl využit ani žádný JavaScript na straně serveru, kterým je třeba *Node.js*[4] či *Rhino*[5]. [17]

Pro formátování struktury HTML dokumentů na straně klienta nebyly použity pouze samotné kaskádové styly, které využívají i šablony *latte*, ale byl využit i dynamický jazyk *LESS*[6], který rozšiřuje CSS o dynamické prvky jako jsou proměnné, mixiny, výpočty nebo dokonce i funkce. [17]

Pro tvorbu kompletního layoutu byl využit moderní *Flexbox Layout*[7], který umožňuje, mimo jiné, například centrování elementů s dynamickou šířkou. Jednou z hlavních předností flexboxu je totiž jistá schopnost vyplnit zbylé místo bez nutnosti tento prostor přepočítat pomocí JavaScriptu. [17]

[2] https://angularjs.org/
[3] https://reactjs.net/
[4] https://nodejs.org/
[5] https://developer.mozilla.org/cs/docs/Mozilla/Projects/Rhino
[6] http://www.lesscss.cz/
[7] http://www.w3schools.com/css/css3_flexbox.asp

Back-end

Návrh a implementace této komponenty, která slouží hlavně ke zpracování dat, navazuje na strukturu webového rozhraní z kapitoly 2.3.

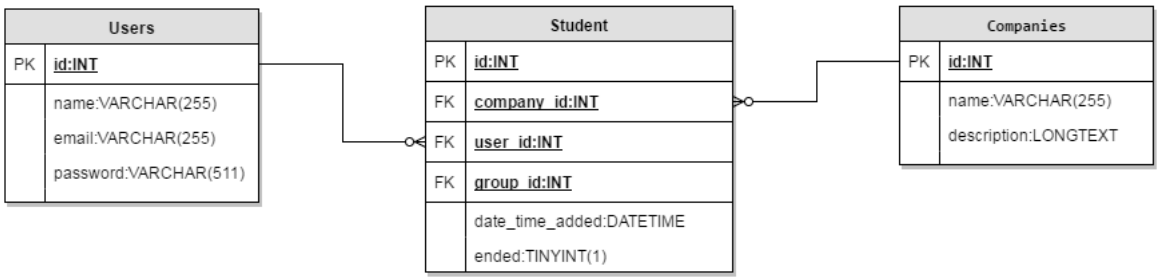
K implementaci serverové části byl využit programovací jazyk *PHP*, který patří v dnešní době mezi nejrozšířenější a zaručuje pro vývojáře velké množství knihoven. Celá aplikace je dále rozdělena na tři oddělené části pomocí návrhového vzoru *Model-View-Controller*, konkrétně na datový model, uživatelské rozhraní a řídící logiku. Pro zrychlení vývoje, a hlavně pro budoucí návaznost projektu, byl také zaveden *PHP framework*. Konkrétně byl zvolen český framework *Nette*, který má k dispozici velké množství hotových open-source pluginů. [17]

Komponenta Back-end je v rámci implementace rozdělena na tři konkrétní moduly. Jedním z nich je modul *Back*, který obsahuje veškeré presentery, ovládání a šablony pro správu autoškol. Druhým je modul *Front*, který má za úkol obstarávat veškeré činnosti žáků. Poslední modul *Admin* slouží ke správě celého webového portálu. [17]

Databáze

Databázová vrstva je zajištěna pomocí *MySQL*, který uplatňuje relační databázový model. V projektu *DoAutoškoly.cz* je využit, místo klasických dotazů SQL, ORM framework *Doctrine 2*[8], který zajišťuje mapování objektů na relační databázi, a to i v opačném směru. Výhodou užití tohoto frameworku je, že lze libovolně přecházet mezi konkrétními typy databází. [17]

Na obrázku 2.1 je znázorněné velmi zjednodušené schéma databáze. Ve schématu je zobrazena pouze nezbytná část, která bude potřebná při návrhu mobilní aplikace pro ověření uživatele.



Obrázek 2.1: Zjednodušené schéma databáze portálu *DoAutoškoly.cz*.

Zabezpečení

Základní zabezpečení je realizováno použitím frameworku *Nette* řešící základní útoky, například *Cross-Site Scriptings* nebo *Cross-Site Request Forgery*[9].

Komunikace mezi serverem a klientem je chráněna před možným útokem pomocí *HTTPS*, konkrétně kryptografickým protokolem *TLS*. Tím je zabráněno odposlouchávání či falšování zpráv. [17]

[8] http://www.doctrine-project.org/
[9] https://doc.nette.org/cs/2.4/vulnerability-protection

Kapitola 3

Cíle projektu

Cílem této diplomové práce je navrhnout a realizovat systém pro on-line záznam jízd autoškoly, konkrétně v návaznosti projektu *DoAutoškoly.cz*, který je blíže popsán v předchozí kapitole 2.

Jedním z cílů této práce je mobilní aplikace pro operační systém *Android*, která bude primárně sloužit pro sběr GPS dat v reálném čase za účelem záznamu jízdy na serveru. Dalším dílčím cílem je rozšíření stávajícího webového portálu, kde uživatel mobilní aplikace bude mít k dispozici prohlížení a úpravu naměřených tras. Z praktického hlediska by bylo ideální, kdyby úprava tras probíhala tam, kde byla vytvořena. Jedním z největších problémů mobilních zařízení je, že mají velmi malou dotykovou obrazovku, která neumožňuje dobrou manipulaci s mapami. Z tohoto důvodu bude možnost úprav pouze na webu.

Prostředníkem mezi mobilní aplikací, webovým klientem a hlavní databází s naměřenými trasami bude serverová aplikace, která zajistí real-time sledování jednotlivých vozidel v rámci konkrétní autoškoly. Dalším důležitým cílem serverové části je „přimknutí“ naměřených polohových dat k reálné vozovce kvůli kvalitnější analýze.

3.1 Motivace

Jak již bylo zmíněno v kapitole 2.2, kvalitativní žebříček autoškol je nyní závislý pouze na slovním a číselném hodnocení od absolventů.

Zkušenosti s nakupováním produktů z internetových obchodů máme všichni, ale málokdo zpětně vyplní hodnocení a recenzi zakoupeného zboží, které nám přijde například na e-mail. Proto si myslím, že tento způsob zpětné vazby je nedostačující, a ne vždy je objektivně vyplněno.

Motivací této diplomové práce je zkvalitnit hodnocení autoškol a tím pádem zvýšit dovednosti samotných řidičů. Důsledkem tohoto snažení by mělo být zlepšení dopravních situací na vozovkách, tedy snížit riziko vzniku dopravních nehod.

Z dřívějších osobních zkušeností vím, že instruktoři si rádi během jízd vyřizují své osobní záležitosti, nechávají studenta čekat na benzinové stanici během konzumace kávy apod. Také se velmi často stává, že žáka navigují do míst, kde jsou očekávané dopravní kolony v centru města. Tento žák potom celou svoji lekci stráví popojížděním a svoje řidičské schopnosti tak příliš nerozšíří. Tyto autoškoly zkrátka dělají vše proto, aby nějakým způsobem ošidily svého zákazníka, ať už finančně nebo z hlediska možnosti sbíraní nových zkušeností.

Zaznamenávání a analýza tras je tedy řešením pro odhalování nepoctivých autoškol a instruktorů. Každý budoucí žák si bude moct jednoduše ověřit a porovnat jednotlivé autoškoly. Tím si dokonale promyslí, komu se svěří do rukou. Zda to bude autoškola, která má v celkovém hodnocení nízký počet projetých křižovatek a často stejné okružní jízdy nebo to spíše bude autoškola, která se snaží žáka seznámit s většinou složitých dopravních situací a komunikací v daném městě.

3.2 Cílová skupina

Cílovou skupinou, kterou by měla tato aplikace oslovit, jsou zejména mladí studenti. Největší skupinou žadatelů o řidičský průkaz jsou lidé ve věku kolem 18 let, kteří většinou stále ještě studují. V prostředí studentů lze snadněji rozšířit informaci o existenci aplikace. Následná soutěživost mezi nimi dokáže vzbudit zájem o aplikaci, která se tímto způsobem bude dále šířit mezi budoucími žadateli o řidičské oprávnění. Pokud by aplikace byla cílena na majitele a jejich lektory, mohlo by dojít opět ke sběru nekorektních dat. Tito lektoři by mohli aplikaci zapnout jen v případě, kdy si lektor je vědom, že data budou korektní a v případě podvodu tuto aplikaci zase vypnout. Z důvodu této obavy jsou cílovou skupinou určeni budoucí studenti autoškoly.

3.3 Spolupráce

Tato diplomová práce probíhá v kooperaci s diplomovou prací Bc. Martina Šouláka na téma *„Systém pro analýzu a vyhodnocení jízd autoškoly“*. Jeho úkolem je analyzovat naměřené trasy studentů a následně prezentovat výsledky na portálu *DoAutoškoly.cz*.

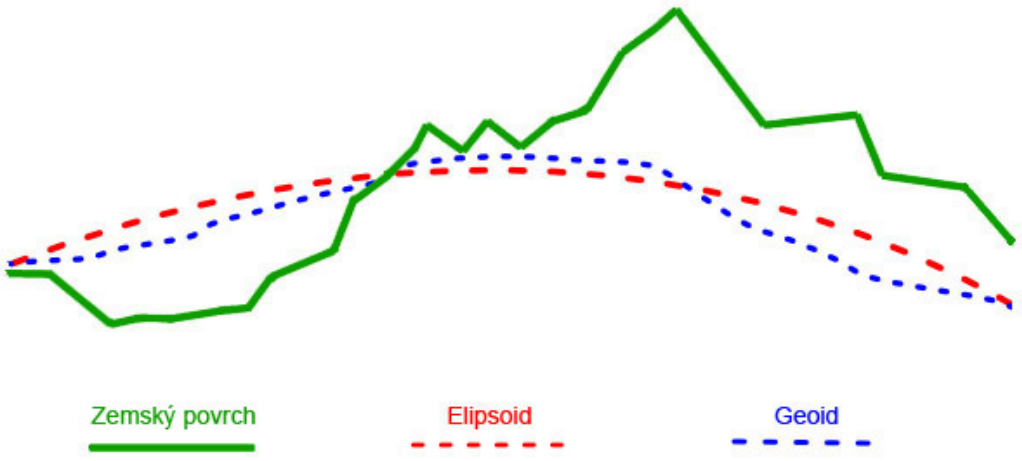
Interference obou prací probíhá na databázové úrovni serveru, která byla konkrétně navržena, realizována a zdokumentována v rámci práce Bc. Martina Šouláka.

Kapitola 4

Geoinformatika

4.1 Model povrchu Země

K přesnému a jednoznačnému určení polohy na povrchu Země je nutný trojrozměrný prostor. Vzhledem k tomu, že tvar naší planety je poněkud členitý a zvrásněný, je téměř nemožné jej jednoduše matematicky popsat. Z tohoto důvodu se zavádí zjednodušení popisu tvaru zemského tělesa. První aproximací zemského povrchu je tzv. *geoid*, který z geodetického hlediska představuje nulovou nadmořskou výšku. [29, 32] Složitost matematického popisu *geoidu* vedla k zavedení referenčního elipsoidu, který lze definovat různými způsoby (volba středu elipsoidu a volba delší a kratší poloosy) a tím volit i různé přesnosti. [18] Na obrázku 4.1 je patrný rozdíl mezi referenčním elipsoidem, vyznačeným červeně, od geoidu, znázorněným modře, v porovnání se skutečným povrchem planety Země, který je vyobrazen zelenou barvou. Je nutné si uvědomit, že výsledná poloha jakéhokoliv bodu (objektu) se vztahuje k elipsoidu, a nikoliv ke skutečnému zemskému povrchu. [18]



Obrázek 4.1: Vztah geoidu a elipsoidu k zemskému povrchu. [29]

4.2 Souřadné systémy

Z důvodu jednoznačného určení polohy jakéhokoli bodu v prostoru vzniká potřeba zavést souřadný systém, ke kterému je případný bod vztažen. Každý souřadný systém musí splňovat tři požadavky: [18]

1. definice polohy je jednoznačná,
2. definice polohy je kvantifikovatelná,
3. lze měřit vzdálenost mezi dvěma objekty.

Vzhledem k návaznosti na konkrétní potřeby se souřadné systémy dělí na dva základní typy: [18, 29]

- **Globální** – Cílem globálních souřadných systémů je obsáhnout celý geografický prostor Země. Jejich velkou výhodou je univerzálnost v popisu planety. Nevýhodou je naopak jistá nepřesnost těchto systémů. Nejrozšířenějšími zástupci jsou *WGS-84* a *UTM*, které se používají zejména pro globální polohovací systémy.
- **Lokální** – Velkou předností lokálních souřadných systémů je přesnost, která je nutná například v katastrálních mapách. Spadá sem například *S-JTSK*, který vznikl transformací náhradního elipsoidu na plochu. V takovém případě lze využít Euklidův vzorec pro výpočet vzdálenosti.

4.3 Souřadné systémy užívané na území ČR

Z hlediska historie bylo na území České republiky a bývalého Československa užíváno několik souřadných systémů, které se od sebe lišily, a to zejména přesností. V současné době jsou závazné na území České republiky tyto geodetické referenční systémy pro zeměměřičské činnosti: [2]

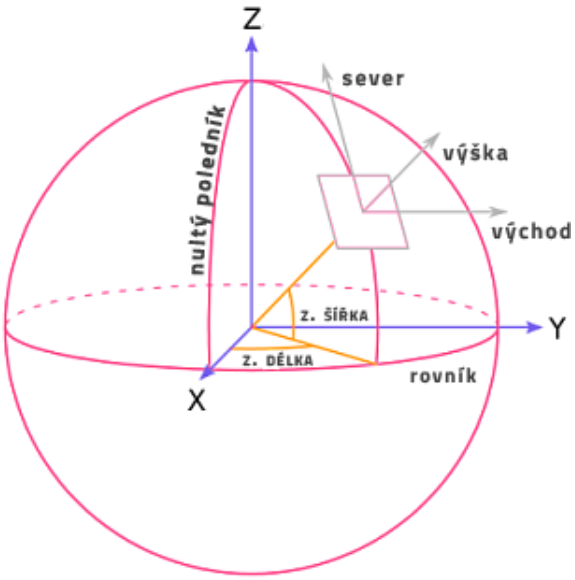
- **WGS-84** – světový geodetický referenční systém 1984,
- **ETRS** – evropský terestrický referenční systém,
- **S-JTSK** – souřadnicový systém Jednotné trigonometrické sítě, katastrální,
- **S-42** – souřadnicový systém, který využívá armáda ČR,
- **Bpv** – výškový systém baltský po vyrovnání,
- **S-Gr95** – tíhový systém 1995.

V následující části textu se budu zabývat pouze souřadným systémem *WGS-84*, který používá jako referenční navigační systém *GPS – NAVSTAR* a je tak nezbytným pro pochopení následující kapitoly.

4.4 WGS-84

V současné době je *WGS-84* nejrozšířenějším globálním souřadným systémem pro popis polohy na povrchu Země, který byl vyvinut ministerstvem obrany USA v roce 1984 pro armádní účely. Později tento systém převzaly členské země NATO. [1]

Jedná se o geocentrický souřadný systém. To znamená, že počátek kartézského souřadného systému je ve hmotném středu Země. Zeměpisná šířka je vztažena k ose y neboli k úhlu, který svírá bod na povrchu Země s rovinou rovníku. Zeměpisná délka je vztažena k ose x neboli k úhlu, který svírá bod na povrchu Země s nultým poledníkem. Osa z je totožná s osou otáčení Země, protíná tedy zemský povrch v bodech jižního a severního pólu. Názorný příklad můžeme vidět na obrázku 4.2. [18, 1]



Obrázek 4.2: Schéma souřadného systému WGS-84. [1]

Kapitola 5

Globální poziční systém – GPS

GPS – NAVSTAR je pasivní družicový systém pro určení polohy na zemi. V současné době je tento systém nejpoužívanější a nejznámější, který je často zjednodušeně označován jako systém GPS. Globální poziční systém využívá souřadnicový systém a referenční elipsoid WGS-84. [5, 13]

Vývoj systému prvotně sloužil pro vojenské účely armády Spojených států amerických, který začal v roce 1973 pod názvem *NAVSTAR*. Během následujících 30 let se zvyšoval potenciál toho systému. V tomto období nastaly největší změny hlavně v rámci kosmického segmentu. [5, 13]

S nárůstem užívání GPS systému pro civilní účely, bylo armádou spojených států rozděleno polohování na dvě přesnosti. Bylo odděleno striktní polohování pro vojenské účely (PPS – *Precision Positioning Service*), kterou mohl využívat jen uživatel s licencí od americké vlády, od civilního využití (SPS – *Standard Positioning Service*) bez nároků na přesné určení striktního bodu. Civilní služba byla k dispozici běžným uživatelům. Ale v devadesátých letech minulého století byla zavedena tzv. selektivní dostupnost, kde pro civilní účely byla přesnost účelně degradována v řádech až na stovky metrů. Teprve až od nového tisíciletí byla selektivní dostupnost deaktivována. [5]

Dnešní globální navigační systémy jsou primárně založeny na obecném principu trilaterace, kde je nutné zabezpečit viditelnost alespoň čtyř satelitů pro 3D fix. Každý satelit musí znát svou přesnou polohu a musíme být schopni dokonale změřit vzdálenost mezi přijímačem a satelitem. [18]

Navigační systém GPS pracuje na principu pasivní komunikace pro určení polohy na zemi. Princip pasivní komunikace spočívá v tom, že družice vesmírného segmentu neustále vysílají NMEA zprávu. Přijímače v uživatelském segmentu tuto zprávu pouze v určitém čase akceptují, nepodílí se tedy na aktivním vysílání dotazu. Z toho vyplývá, že při pasivní komunikaci satelit nezná polohu uživatele. [18]

5.1 Segmenty GPS

Globální poziční systém se skládá ze tří segmentů: [18, 5]

- **Space segment** – Kosmický segment je tvořený sítí družic. Globální poziční systém se sestává z 24 satelitů na šesti oběžných stacionárních drahách ve výšce asi 20200 km se sklonem 55 ° od roviny rovníku. Vzájemně jsou posunuty o 60 ° a oběžná doba jedné družice je asi 11 hodin a 58 minut. To nám zaručuje, že z každého místa na Zemi lze vidět 5 až 8 satelitů. [15] Pro příjem povelových signálu a navigačních zpráv

je v družici umístěna komunikační jednotka. Každá družice dále obsahuje palubní počítač, baterii a atomové hodiny, jejichž přesnost dosahuje 10^{-13} s. [13]

- **Control segment** – Prioritním úkolem řídícího segmentu je monitorovat data z jednotlivých družic a následně je vyhodnocovat. Jednotlivé monitorovací stanice jsou rozmístěny po celém světě a obsahují svoje vlastní přijímače s atomovými hodinami. Díky tomu kontrolují pseudovzdálenosti mezi kontrolními středisky a jednotlivými satelity. Další schopností je synchronizovat atomové hodiny všech satelitů. [13]
- **User segment** – Uživatelský segment se skládá z přijímače a uživatelského prostředí. Přijímač pomocí obdržených informací od jednotlivých družic, zejména jejich čas z atomových hodin a aktuální rychlosti, dokáže určit polohu. [5] Přijímač vyžaduje viditelnost minimálně čtyř satelitů pro získání přesné 3D polohy. Obecně platí, čím více viditelných družic, tím přesnějším souřadnice. [18]

5.2 NMEA

Pro komunikaci přijímačů a družic byl vyvinut protokol NMEA 0183, který je interpretovaný jako řetězec složený z ASCII znaků. V NMEA standardu existuje mnoho datových vět pro různá zařízení. Pokud je zpráva spojena se zpracováním signálu z GPS, pak prefix zpráv je „GP“. Minimální doporučená informace pro navigaci je nesena větou *RMC*. Následující zobrazený příklad 5.1 je popsán v tabulce 5.1. [5]

$$GPRMC, 235600.833, A, 4918.0139, N, 1631.8216, E, 0.08, 14.62, 140316, 0.01, E * 25 \quad (5.1)$$

	Formát	**Příklad**	**Význam**
1	hhmmss.sss	235600.833	aktuální čas (UTC)
2	c	A	status věty (A = platná, V = neplatná)
3	ddmm.mmmm	4918.0139	zeměpisná šířka
4	c	N	indikátor sever/jih (N = sever, S = jih)
5	ddmm.mmmm	1631.8216	zeměpisná délka
6	c	E	indikátor východ/západ (E = východ, W = západ)
7	d.d	0.08	vodorovná rychlost v uzlech
8	d.d	14.62	kurz pohybu ve stupních
9	ddmmyy	140316	aktuální datum
10	d.d	0.01	magnetická deklinace ve stupních
11	c	E	indikátor východ/západ (E = východ, W = západ)
12	*xx	25	kontrolní součet

Tabulka 5.1: Věta NMEA. [5]

5.3 Princip získání polohy

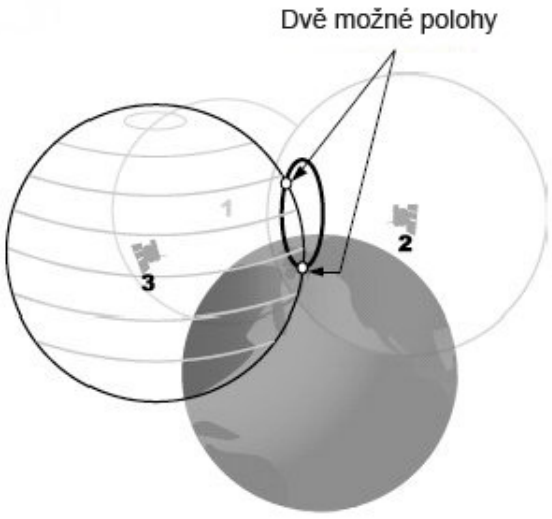
Pro přesné určení polohy musí vesmírný segment, blíže popsaný v kapitole 5.1, zaručit potřebné pokrytí všech míst na Zemi tak, abychom mohli vypočítat polohu s dostatečnou přesností. Pro zjištění naší pozice je nutné znát naprosto přesnou polohu viditelných satelitů, které se pohybují konstantní rychlostí po oběžné dráze. Princip vypočítání polohy je založen na takzvané *trilateraci*.

Princip 2D trilaterace spočívá ve znalosti vzdáleností od tří bodů, u kterých je známá přesná pozice. Poté se jedná o triviální výpočet o soustavě tří rovnic o dvou neznámých vyplývající z Pythagorovy věty. Výsledná pozice je průsečíkem tří kružnic. [18]

3D trilaterace má prakticky totožný princip jako výše zmíněná 2D trilaterace, ale s tím rozdílem, že vizualizace probíhá na třech protínajících se koulích, jako je znázorněno na obrázku 5.1. Známe-li vzdálenost od tří satelitů z výpočtů doby šíření signálu k přijímači a připočítáme-li Zemi jako čtvrtou protínající se kouli, lze již snadno vypočítat požadovanou pozici. V tomto případě je řešení soustavy rovnic 5.2 dvojí, ale pouze jedno se může nacházet na povrchu Země, které získáme eliminací. [28] Funkce d_i v rovnici 5.2 značí známou vzdálenost mezi satelitem S_i a vypočítávanou pozicí P. Proměnná S_{i_a} zastupuje souřadnici satelitu S_i, kde a je osa v trojrozměrném kartézském souřadném systému. Proměnná P_b zastupuje vypočítávanou pozici přijímače, kde b je osa v trojrozměrném kartézském souřadném systému.

Přijímače z pravidla požadují čtyři a více satelitů, aby zvýšily přesnost, a navíc získaly i znalost nadmořské výšky.

$$d_i(S_i, P) = \sqrt{(S_{i_x} - P_x)^2 + (S_{i_y} - P_y)^2 + (S_{i_z} - P_z)^2} \tag{5.2}$$



Obrázek 5.1: Princip 3D trilaterace. [12]

Kapitola 6

Android

Android je v současné době nejpoužívanějším operačním systémem chytrých telefonů. Hlavní oblastí uplatnění systému *Android* jsou obecně zařízení s malými obrazovkami, které většinou nemají ani hardwarovou klávesnici, jsou jimi „chytré telefony“. [3]

Výhody těchto zařízení jsou zřejmé. Máme neustálé spojení s okolním světem pomocí snadného přístupu k internetovým službám a trend těchto zařízení neustále roste, zejména u mladých lidí. Na druhou stranu tento fakt má také svou daň. Ve zkratce řečeno, telefony jsou snad ve všech směrech malé, včetně obrazovky, klávesnice a polohovacího zařízení, a to stejné platí dokonce i o procesorech.

Neopominutelný fakt je, na který nesmíme zapomenout při vývoji *Android* aplikace, že výsledný produkt bude běžet právě na mobilním telefonu. Budoucí uživatel by asi nebyl příliš rád, kdyby aplikace vytěžovala procesor natolik, že by telefon ani nezaznamenal příchozí hovor nebo kdyby v nouzové situaci nemohl nikam zavolat.

6.1 Verze Androidu

Android v současné době vlastní společnost Google Inc., ale vždy tomu tak nebylo. Google odkoupil společnost *Android Inc.* v roce 2003 a od té doby neustále roste a vyvíjí se. [3]

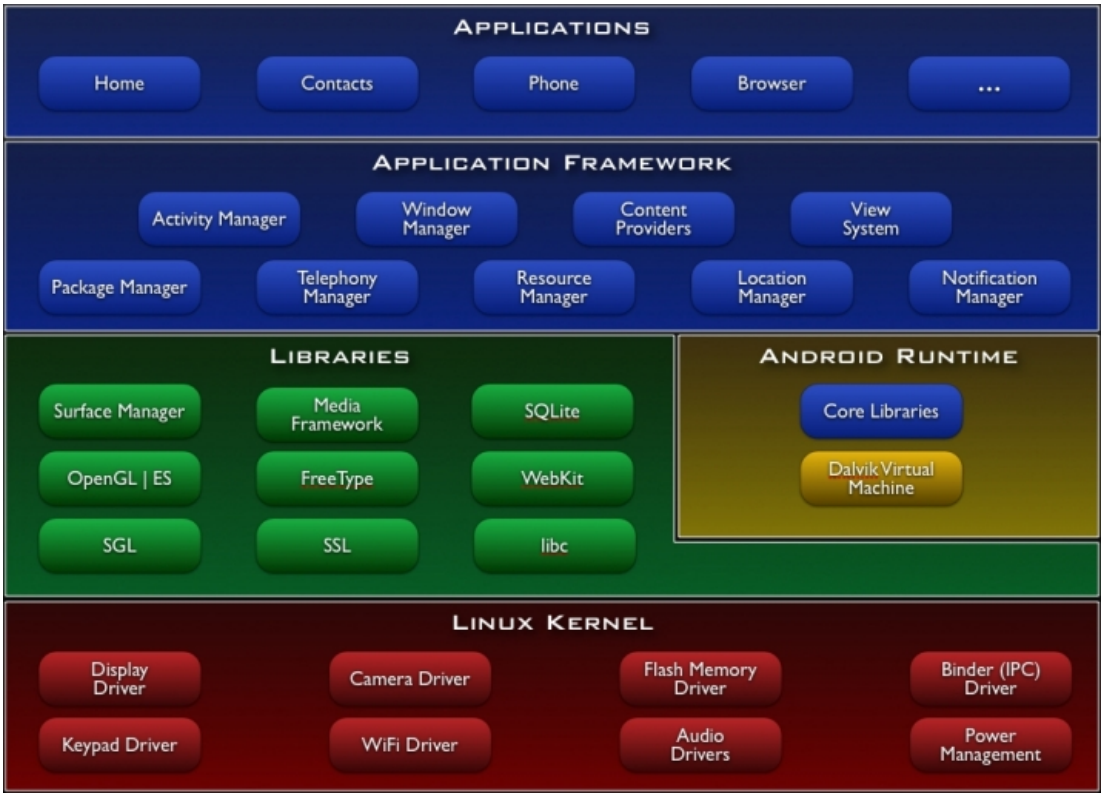
Jednotlivé verze operačního systému *Android* mají své číselné označení. Zajímavostí je, že verze mají i kódové označení v podobě zákusku, které jsou navíc v abecedním pořadí. V době psaní této diplomové práce Google uvolnil jako poslední API 24 s kódovým názvem *Nougat*, ale prozatím nejpoužívanější verzí stále zůstává API 19 s kódovým názvem *KitKat*. [21, 8]

Při vývoji *Android* aplikace je velmi důležité si na začátku promyslet, jakou nejstarší verzi podporovat. Vezmeme-li v potaz, že životnost mobilních telefonů je zhruba dva, maximálně tři roky, tomu odpovídá již zmíněná verze *KitKat*, která má zastoupení 25 %, ale předchozí verze již rapidně klesají. Poslední verze, kterou se má smysl zabývat je *Jelly Bean*, která má v součtu zastoupení 12 %. [8]

6.2 Architektura

Před samotným vývojem aplikací je vhodné si ujasnit principy architektury operačního systému *Android*.

Operační systém *Android* se skládá z několika základních částí, které můžeme vidět na obrázku 6.1. Jednotlivé vrstvy budou dále popsány od nejnižší architektonické vrstvy: [21]



Obrázek 6.1: Architektura platformy *Android*. [20]

- **Jádro (Linux Kernel)** – Upravené linuxové jádro tvoří základ operačního systému a slouží primárně k interakci s hardwarem zařízení. Jádro zabezpečuje ovladače zařízení, síťové připojení, přístup ke zdrojům, správu procesů, apod.

  Jednotlivé aplikace a služby jsou spouštěny v samostatných procesech. Problémem je, že často mezi sebou potřebují komunikovat. Proto jako zprostředkovatel existuje tzv. *Binder*, který s využitím sdílené paměti dokáže komunikovat mezi konkrétními procesy, pokud jsou zaregistrované ve službě *Service Manager*. V systému *Android* je meziprocesová komunikace pro usnadnění řešena pomocí *AIDL*[1].

- **Knihovny (Libraries)** – Poskytují přímý přístup ke komponentám systému. Tvoří mezivrstvu mezi linuxovým jádrem a komponentami vyšších vrstev. Spadají sem například knihovny pro práci s grafikou nebo knihovna *SQLite*.

- **Android Runtime** – Slouží primárně pro běh aplikací, kde každá aplikace je vedena jako samostatný proces, který využívá vlastní instanci virtuálního stroje *Dalvik Virtual Maschine*.

- **Aplikační framework (Application Framework)** – Je vrstva obsahující další knihovny, které jsou napsané v *Javě*. Obsahují opakovaně použitelný software. Jedná se zejména o grafické a ovládací prvky.

[1] https://developer.android.com/guide/components/aidl.html

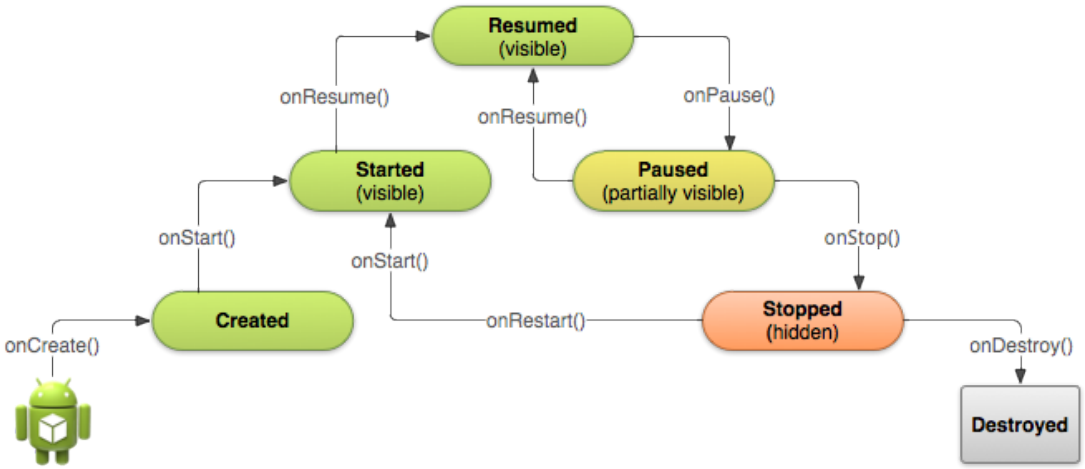
- **Aplikace (Applications)** – Je nejvyšší vrstva architektury, na které jsou již konkrétní aplikace. Systém *Android* považuje všechny aplikace za rovnocenné a přiděluje jim stejné prostředky.

6.3 Základní součásti aplikace

Aktivity (Activity)

Aktivity jsou základními stavebními bloky uživatelského rozhraní s krátkým životním cyklem. Každá aplikace může sestávat z více aktivit, které si navíc můžou mezi sebou předávat informace. Při spuštění aplikace se vždy zobrazí na popředí jedna z aktivit. Platforma *Android* je navržena tak, aby zvládala velké množství nenáročných aktivit, které zobrazují uživatelské rozhraní a zachytávají interakce s uživatelem. Díky tomu je možné mezi jednotlivými aktivitami libovolně přecházet. [3, 21]

Každá aktivita, jak můžeme vidět na obrázku 6.2, během svého životního cyklu prochází různými stavy: [7, 21]



Obrázek 6.2: Životní cyklus aktivity. [7]

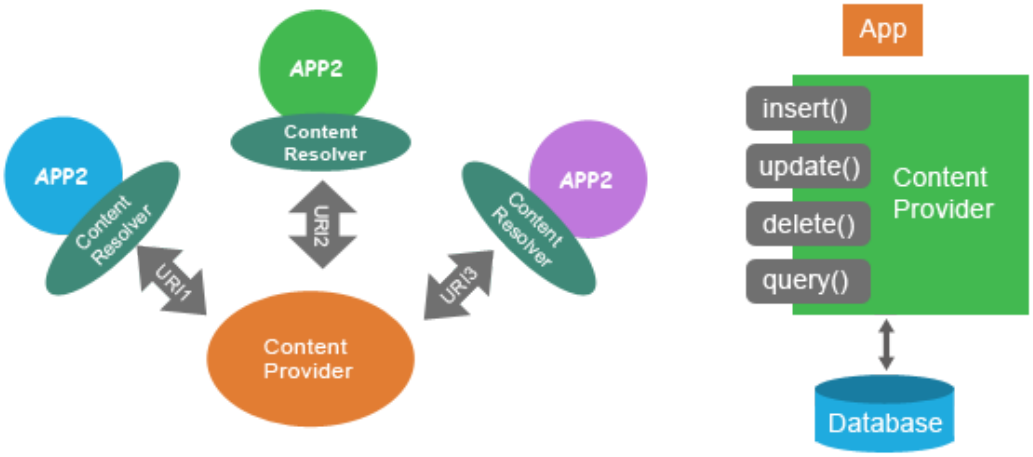
- **onCreate()** – vždy při prvním spuštění aktivity, ale aktivita je zde stále neviditelná, z pravidla se zde zavádí *layout* a inicializují se prvky *view*,
- **onRestart()** – poté, kdy je aktivita stopnuta a přechází znovu do popředí k uživateli,
- **onStart()** – vždy, kdy aktivita přechází do popředí k uživateli,
- **onResume()** – právě začala interakce s uživatelem a aktivita je na vrcholu zásobníku,
- **onPause()** – v momentě, kdy je aktivita odebrána z vrcholu zásobníku, například kvůli jiné spuštěné aktivitě,
- **onStop()** – v momentě, kdy aktivita není dlouhodobě viditelná uživateli,
- **onDestroy()** – při ukončení životnosti aktivity.

Služby (Services)

Na rozdíl od aktivit, které mají krátký životní cyklus, jsou služby navrženy k neustálému provozu a také běží na pozadí bez uživatelského rozhraní. Tyto služby se využívají například k přehrávání hudby na pozadí, nepřetržitému získávání polohy pomocí GPS nebo ke kontrole aktualizací v informačním kanálu. To všechno i v případě, kdy aktivita, která službu spustila, již není na popředí nebo dokonce byla vypnuta. Služby také dokážou zpřístupnit rozhraní dalším procesům v zařízení, které se registrují pomocí *Service Manager*. [3, 21]

Poskytovatelé obsahu (Content providers)

Poskytovatelé obsahu umožňují ukládání a také sdílení dat mezi více aplikacemi a procesy. Aplikace tak mohou přistupovat, pokud znají patřičnou *Content Uri*[2], k informacím ostatních aplikací, které jsou ovšem vedené jako poskytovatelé obsahu. Naše aplikace tak například může přistupovat k výpisům hovorů nebo SMS. Poskytovatelem obsahu může být téměř cokoliv, ale většinou se jedná o interní databázi platformy *Android*, tedy *SQLite* databázi. [3, 21]



Obrázek 6.3: Poskytovatel obsahu. [19]

Záměry (Intents)

Záměry jsou systémové zprávy, které upozorňují aplikace na vzniklé události. Jedná se například o změnu hardwarového nastavení, změnu dat, příchozí data nebo samotné události aplikací. Na tyto záměry lze posléze patřičně zareagovat. [3, 21]

Broadcast Receivers

Fungují v telefonu jako přijímače naslouchající na pozadí a reagují na předem registrované typy událostí, které se odehrávají v zařízení (již zmíněné Záměry). Kupříkladu se může jednat o zprávu při změně stavu připojení k internetu nebo při zapnutí a vypnutí displeje zařízení. [21]

[2] https://developer.android.com/reference/android/content/ContentUris.html

6.4 Databáze SQLite

Databáze *SQLite* je vestavěnou databází všech zařízení, které jsou založené na operačním systému *Android*. [3]

SQLite je vlastně pouze obyčejná knihovna, která implementuje relační databázový systém. Není založena na principu klient-server a ani není spuštěna jako samostatný proces, jako například *MySQL*. Naopak umožňuje otevřít datový soubor, se kterým umí pracovat prostřednictvím jazyka *SQL*. [31]

Databáze *SQLite* má pouze omezené datové typy, které jsou: [31]

- **NULL** – hodnota null,
- **INTEGER** – celočíselná hodnota,
- **REAL** – reálná hodnota,
- **TEXT** – textový řetězec,
- **BLOB** – binární data.

6.5 Aplikační rozhraní pro získání polohy

Platforma *Android* nabízí podle oficiální dokumentace řadu systémových služeb, mezi niž patří i třída `LocationManager` poskytující přístup ke službě, která aplikacím umožňuje pravidelné získávání zeměpisné polohy přístroje nebo vytvoření záměru v okamžiku, kdy se zařízení nachází v blízkosti konkrétní zeměpisné oblasti. [10]

Aktivita nebo služba, ve které chceme zajistit získávání geografické polohy, musí mít implementované rozhraní `LocationListener` používající se právě pro přijímání oznámení od `LocationManager` v momentě, kdy dojde ke změně polohy. Aby byly metody řádně volány a tím docházelo i k opakovanému určení aktuální lokace, je nutné `LocationListener` registrovat u správce služby pomocí metody `requestLocationUpdates`. Tato funkce umožňuje vývojářům zvolit konkrétního poskytovatele (internet nebo GPS) a také minimální změnu vzdálenosti v metrech a času v milisekundách, kterou je nutné pro aktualizaci překonat. Podobně, lze metodou `removeUpdates` naslouchání i odebrat. [9]

6.6 Lokalizace v systému Android

Volba zdrojů, pomocí kterých lze v systému *Android* získat polohu, je poměrně velká. Téměř v každém zařízení existují minimálně čtyři zdroje pro získání aktuální polohy: GPS, Wi-Fi Positioning System (WPS), GSM Cell ID a senzory. Vzhledem k této práci budou dále popsány pouze první dvě metody, neboť nemůžeme říci, že by uživatelé mobilní aplikace měli k dispozici ve vozidle Wi-Fi připojení. Naopak senzory poskytují pouze informaci o orientaci, míry rotace zařízení apod.

Důležitým kritériem při výběru zdrojů pro získání polohy je míra přesnosti a pokrytí, ale také i průměrná spotřeba energie. Bohužel dokumentace systému *Android* je z hlediska využití baterie nedostatečná a neudává přesné hodnoty. Pouze uvádí, že GPS je z pohledu spotřeby energie náročnější než v případě síťového připojení, a to zejména při počátečním získání polohy. [11]

Globální Poziční Systém

Jak jsem již uvedl v kapitole 5, z každého místa na Zemi lze vidět 5 až 8 satelitů, které poskytují signál, mimo jiné, s aktuální polohou satelitu a časové razítko. GPS přijímač umístěný v zařízení se systémem *Android* dokáže vypočítat dobu šíření signálu a tím pádem i vzdálenost mezi přijímačem a satelitem. Pro vypočítání přesné 3D polohy potřebuje zařízení viditelnost nejméně čtyř satelitů.

- **Pokrytí** – Sice máme zaručenou jistou viditelnost satelitů po celé Zemi, ale i přesto se jedná o nejnáchylnější způsob získání polohy. To, že GPS špatně funguje v uzavřených prostorách, by ani tak nevadilo vzhledem k využití v této práci. Musíme si ovšem uvědomit, že GPS přijímače v mobilních zařízeních jsou levného charakteru a jsou tedy velmi náchylné na počasí nebo na velké objekty v blízkosti přijímače. [5]
- **Přesnost** – Z hlediska přesnosti je využití GPS nejvýhodnější. V ideálním stavu lze získat přesnost vyšší než 3 metry, při optimálních podmínkách je ale nutné počítat s přesností spíše 3 až 10 metrů. V závislosti na špatných podmínkách dochází ke zhoršení přesnosti pozice v řádech až desítek metrů. Nevýhodou vysoké přesnosti je také fakt, že uživatelé polohu nezískají tak rychle, jak by očekávali. Takzvaný *time to first fix* (TTTF) může trvat až 10 minut. [5, 36]

GSM Cell-ID

U lokalizace pomocí sítě dochází k žádosti o získání polohy z mobilního zařízení směrem do sítě, tedy naopak než u GPS, kde je využit pasivní přístup. Mobilní zařízení zpětně přijme informaci o poloze. V tomto případě dochází ke zvyšování zatížení přenosu dat po síti a tím pádem i spotřeby energie zařízení. [36, 27]

- **Pokrytí** – Výhodou tohoto typu zjišťování polohy je pokrytí i v uzavřených prostorách, což zaručuje možnost daný objekt neustále sledovat v reálném čase. Podobně jako u GPS se jedná o globální pokrytí. [27]
- **Přesnost** – Přesnost není logicky tak vysoká jako v případě GPS, která u tohoto síťového připojení závisí zejména na velikosti buňky. Čím menší buňky v dané oblasti máme, tím získáme větší přesnost. Problémem je, že vyšších přesností se přiblížíme pouze v rámci větších měst. Mimo města se můžeme dočkat přesnosti až stovek metrů. [36]

Kapitola 7

Komunikace v reálném čase

V této části práce budou čtenáři nastíněny základní principy komunikace v reálném čase vzhledem k webovému klientu a aplikaci pro mobilní platformu *Android*. Budou zde nastíněny technologie, které lze využít zároveň na serverové a klientské straně v prostředí webu a chytrého telefonu.

Klasická webová aplikace pracuje na principu *klient-server*. Klient pošle požadavek na server, který po zpracování dat zašle zpět odpověď. To znamená, že komunikace je vždy zahájena ze strany klienta. Nové informace se u klienta neobjeví hned, ale vždy při zaslání nového požadavku. Tento způsob se také označuje jako *Pull*.

Naopak komunikace v reálném čase může být v praxi zahájena z kterékoliv strany. Hlavní výhodou takové aplikace je aktuálnost informací. Uživatel může sledovat „živě“ například pohyb vozidel na mapovém podkladu, a to i v případě, že se jedná o více uživatelů v jednom okamžiku. *Push* princip spočívá v neustále otevřeném spojení mezi klientem a serverem, které je postavené na protokolu *HTTP*.

Dalším způsobem je vytvoření přímého *TCP/IP* spojení pomocí *socketů*. Zde se nabízí využití protokolu *WebSocket* vytvořeným konsorciem *W3C*, který bude blíže popsán v následující části této kapitoly.

7.1 Technologie

Polling

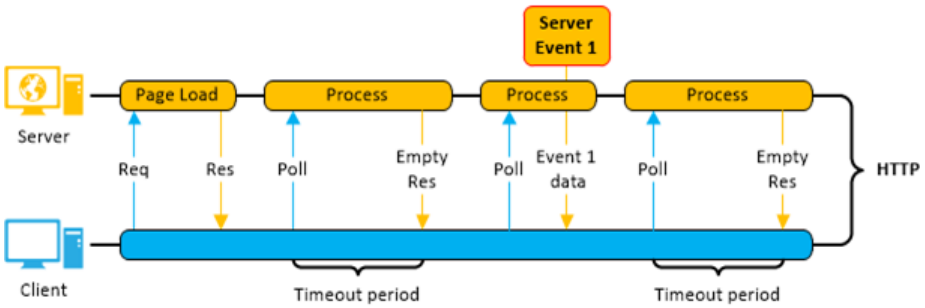
Polling je nejjednodušší způsob implementace spadající pod princip *Pull* spočívající v opakujících se požadavcích s cílem získání nových dat od serverové aplikace. Problémem je vysoká zátěž serveru, která je způsobená velkým počtem příchozích požadavků v krátkém intervalu, které navíc mají často negativní odpověď. [30, 22]

Long polling

Technika *Long polling* kombinuje předešlou techniku s persistentním spojením `keep-alive` a využívá pro komunikaci mezi klientem a serverem objekt `XMLHttpRequest`, zkráceně *XHR*. Tato technika spadá pod princip *Push*. [30, 22]

Long polling (obrázek 7.1) vytvoří požadavek *XHR* a server následně potvrdí spojení s klientem. Aby spojení zůstalo stále otevřené, server nepošle žádnou odpověď do doby, dokud nejsou k dispozici požadovaná data. V případě, že nová data jsou k dispozici, server

je pošle klientovi a následně zruší spojení. Klient je poté nucen vytvořit další nové spojení pomocí protokolu *HTTP*. [30, 22]



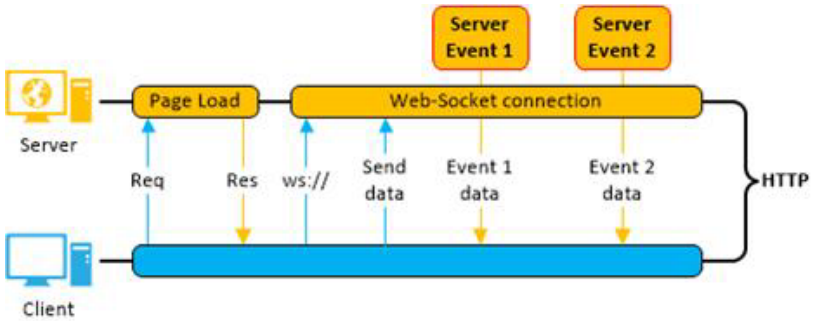
Obrázek 7.1: Otevření komunikace pomocí *Long polling*. [35]

WebSocket

WebSocket je poměrně mladá webová technologie příchozí se specifikací *HTML 5*, která umožňuje obousměrnou komunikaci mezi klientem a serverem pomocí *TCP/IP*. Protokol *WebSocket* byl společně s aplikačním rozhraním standardizován v roce 2011 pod záštitou *IETF* v rámci *RFC 6455*[1].

Rozhraní *WebSocket* definuje plně duplexní komunikační kanál, který funguje na základě jediného *socketu* v rámci celého webu. *Sockety* nabízí enormní snížení zatížení serveru, oproti předchozím výše popsaným metodám, které se tento princip snažily různými způsoby pouze simulovat. [34]

Spojení mezi klientem a serverem, tzv. „*handshaking*“ (obrázek 7.2), začíná strana klienta požadavkem *HTTP* se speciálním parametrem `Upgrade: websocket` v hlavičce. Tento výraz informuje server, že klient si přeje navázat spojení pomocí protokolu *WebSocket*. Pokud server tento protokol podporuje, vrátí odpověď se stavem `101 Switching Protocol`. Následně, když je *handshaking* dokončen a počáteční připojení pomocí protokolu *HTTP* je nahrazeno protokolem *WS* pro nezabezpečenou komunikaci nebo *WSS* pro zabezpečenou komunikaci, může kterákoliv strana zahájit další delegaci zpráv. [22, 26]



Obrázek 7.2: Otevření komunikace pomocí *WebSocket*. [35]

[1] https://tools.ietf.org/html/rfc6455

7.2 Srovnání technologií

Jak již bylo uvedeno výše, *HTML5 WebSocket* poskytuje dramatické zlepšení událostmi řízené webové aplikace zejména při komunikaci v reálném čase, na rozdíl od předchozích metod, které se snažily pouze simulovat plně duplexní připojení ve webovém prohlížeči.

Řada metod, které jsou založené na dlouhotrvajících požadavcích *HTTP*, mohou mít často existenční problémy. Jedním problémem je právě délka spojení, kde klientský prohlížeč může mít omezení na počet současných dotazů na konkrétní doménu. Dalším problémem je, že prohlížeč může sám ukončit spojení v případě, kdy po určitou dobu nepřijde žádná odpověď. [24]

Problém může dokonce nastat i na cestě mezi klientem a serverem, nejčastěji v podobě firewallů a proxy serverů, které sami mohou ukončit dlouhotrvající nebo z jejich pohledu nevyužitá spojení. [24] Technologie *WebSocket* tímto problémem netrpí. Naopak svým návrhem se snaží těmto problémům předcházet. Spojení je sice vytvářeno pomocí *HTTP* protokolu, ale s vytvořením spojení je „povýšeno“ na protokol *WS*. Součástí této technologie je jistá detekce proxy serverů a případné automatické nastavení tunelů příkazem `connect`, který nastaví *TCP/IP* spojení s určitým hostitelem a portem. Jakmile je tunel nastaven, může komunikace dál proudit bez zábran. [34, 25]

Nevýhodou *WebSocket* může být fakt, že se jedná o novou technologii a potřebuje ke svému provozu rozhraní na straně webového prohlížeče. Uživatelé se staršími prohlížeči mohou být odříznuti od nově vytvořené *real-time* aplikace. Tento problém řeší frameworky, které implementují minimálně výše zmíněné technologie. Vývojář si může vybrat z aplikačních rámců, jako jsou například *SocketIO*[2], *XSockets*[3], *SignalR*[4], komerční služby *Pusher.com*[5] a další, podle volby prostředí a implementačního jazyka.

[2] https://socket.io/
[3] https://xsockets.net/
[4] http://signalr.net/
[5] https://pusher.com/

Kapitola 8

Návrh řešení aplikace klient-server

Kapitola 2 uvedla čtenáře do problematiky autoškol v České republice a následující 3. kapitola vytyčila základní cíle pro tuto práci. Pro upřesnění je důležité zdůraznit, že prioritním cílem je navrhnout a následně implementovat systém pro on-line záznam jízd autoškol v návaznosti s diplomovou prací Bc. Martina Šouláka, jejímž hlavním cílem je realizovat systém pro analýzu a vyhodnocení jízd autoškol. Střet zájmů obou prací je na úrovni databázové vrstvy, která je realizována a zdokumentována v práci Bc. Martina Šouláka.

Prvotní myšlenkou bylo vytvoření jedné serverové aplikace, která měla zabezpečit komunikaci s mobilními klienty v reálném čase za účelem sběru geografických dat a následně tyto data analyzovat a vyhodnotit pro účely portálu *DoAutoškoly.cz*. V rámci tohoto systému měla být k dispozici pouze jedna databáze pro ukládání geodat. Prioritně je potřeba zabezpečit rychlou a spolehlivou komunikaci mezi serverem a mobilním klientem. Současně musí server komunikovat i s webovým klientem kvůli sledování vozidla v reálném čase. Z toho vyplývá, že řešení zahrnující jednu serverovou aplikaci s jednou databází je krajně nevyhovující, protože by musela současně zvládnout zpracovat a analyzovat velké množství geografických dat, jejichž prostorové operace jsou velmi náročné. Z tohoto důvodu je žádoucí, aby byla analýza a sběr geografických dat vzájemně oddělena.

Diplomová práce je funkčně rozdělena na tři části, konkrétně na mobilní aplikaci pro platformu *Android*, webového klienta pro majitele a studenta autoškoly a v neposlední řadě serverovou aplikaci zaštiťující komunikaci mezi nimi. Tyto součásti celého systému budou v následujících podkapitolách rozebrány a podrobně navrhnuty.

8.1 Návrh struktury systému pro on-line záznam jízd

Návrh struktury zabezpečující on-line záznam jízd je popsán pomocí schématu na obrázku 8.1.

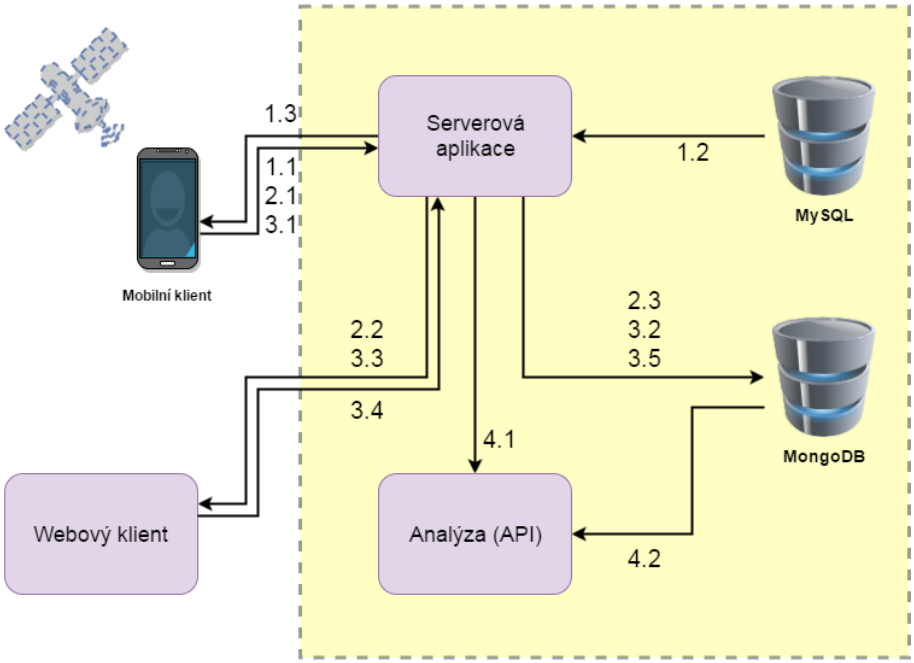
V systému figuruje hned několik databází pro ukládání dat autoškol a jejich studentů. Jak již bylo zmíněno v kapitole 2, současný portál disponuje relační databází *MySQL*, která není příliš vhodná pro ukládání geografických dat, ale spíše slouží výhradně pro autentizaci uživatelů a umožňuje přístup k seznamu autoškol v České republice. Druhou databází ve struktuře systému je *MongoDB*[1], spadající do kategorie *NoSQL*[2], ke které bylo poskytnuto aplikační rozhraní formou modulu. Jejím hlavním existenčním důvodem je zajistit uložení

[1] https://www.mongodb.com/
[2] https://www.mongodb.com/nosql-explained

geodat a kompletních naměřených jízd uživatelů na serveru. Třetí a poslední databáze celého systému *DoAutoškoly.cz* slouží primárně pro analýzu dat. Na obrázku není již vyobrazena, protože nesouvisí přímo s návrhem pro účely této diplomové práce, ale je součástí návrhu aplikačního rozhraní pro analýzu jízd autoškoly v práci Bc. Martina Šouláka.

Na obrázku 8.1 znázorňující schéma navrhovaného systému je pro zjednodušení zobrazen pouze jeden mobilní a webový klient. V reálném použití je samozřejmostí, že komunikace může být zprostředkována mezi více zařízeními současně, vyjímaje okrajových situací, které budou popsány v kapitole 10 zabývající se implementací serverové a klientské části celého systému pro on-line záznam jízd.

Proudění dat mezi komponentami je znázorněno za pomoci orientovaných šipek. Konkrétní komunikace mezi jednotlivými částmi systému jsou rozděleny do několika fází.



Obrázek 8.1: Schéma navrhovaného systému pro on-line záznam jízd.

První fáze – autentizace uživatele

První fází, značené na obrázku jako `1.x`, je autentizace dříve registrovaného uživatele pomocí webového portálu *DoAutoškoly.cz*. Mobilní zařízení komunikuje vždy pouze se serverovou aplikací (`1.1`). Ta provede pomocí databáze *MySQL* ověření uživatele (`1.2`) a zpět pošle patřičná data o studentovi (`1.3`). Žákovi autoškoly musí být umožněn i off-line záznam tras, proto identifikace uživatele zůstane uchována v telefonu po celou dobu existence aplikace v zařízení nebo do ručního odhlášení uživatelem.

Druhá fáze – sběr dat v reálném čase

Přihlášený uživatel může v další fázi (`2.x`) zaznamenávat svoji jízdu v reálném čase. V případě povolení pouze off-line záznamu je tato fáze přeskočena. Mobilní zařízení odesílá (`2.1`) serveru svoji aktuální polohu bod po bodu reprezentovanou souřadnicemi šířka a délka společně s časovým razítkem. Server následně přijatou souřadnici pošle

přihlášenému webovému klientovi sledujícího vozidla dané autoškoly (2.2) a zároveň ji uloží do databáze *MongoDB* (2.3).

Třetí fáze – uložení a optimalizace dat

Po dokončení záznamu trasy, mobilní zařízení pošle kompletní jízdu serverové aplikaci (3.1), která provede optimalizaci dat v podobě „přimknutí“ jednotlivých souřadnic k reálné vozovce na mapě a následně aktualizovanou trasu uloží do databáze (3.2).

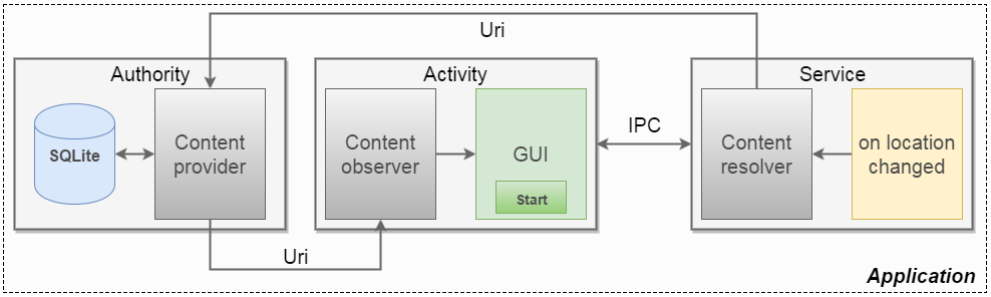
Každý uživatel, který úspěšně zaznamenal svoji jízdu, má v případě nekorektně naměřených dat právo editovat své trasy. V takovém případě serverová aplikace pošle webovému klientovi celou trasu (3.3), který ji po dokončení editace opět pošle zpět serveru (3.4) a ten ji uloží do databáze *MongoDB* (3.5).

Čtvrtá fáze – předání dat

Po dokončení cvičné jízdy autoškoly nebo po úspěšné editaci trasy, serverová aplikace pomocí aplikačního rozhraní deleguje zprávu informující o existenci nových dat (3.5). Serverová aplikace zajišťující analýzu provede přenos nových nebo aktualizovaných dat do své interní databáze (4.2) a následně je vykonáno vyhodnocení tras.

8.2 Návrh struktury Android aplikace

Návrh struktury mobilní aplikace s operačním systémem *Android* je popsán schématem z obrázku 8.2. Vzhledem k faktu, že mobilní zařízení má sloužit primárně pro sběr geografických dat, bude při návrhu aplikace zohledněna právě tato fáze.



Obrázek 8.2: Schéma navrhované struktury *Android* aplikace.

Jednou z možností, jak implementovat sběr dat, je vytvoření aktivity se speciálním vláknem, který bude obsluhovat aplikační rozhraní pro automatické a opakované získání polohy (kapitola 6.5). Tento přístup nezaručuje dlouhodobý běh aplikace, protože platforma *Android* kontroluje a následně ukončuje z jejího pohledu nepotřebné aktivity. Pro tyto účely existují, podle kapitoly 6.3, tzv. služby, které ovšem nevlastní grafické uživatelské rozhraní a je tak nutné zajistit komunikaci právě s touto službou.

Navržená aplikace se skládá ze tří hlavních částí a při spuštění má každá z nich přidělen svůj vlastní proces. Konkrétně se jedná o službu (`Service`), aktivitu (`Activity`) a autoritu (`Authority`). Autorita pomocí *Content Uri* odkazuje na celý `ContentProvider`, který poskytuje přístup k databázi a je to tedy způsob, jak sdílet data mezi procesy. Pro komunikaci mezi procesy je nutné implementovat meziprocesovou komunikaci (*IPC*). Třídy `ContentProvider` a `ContentResolver`, které mají dokonce stejné rozhraní, již tento typ

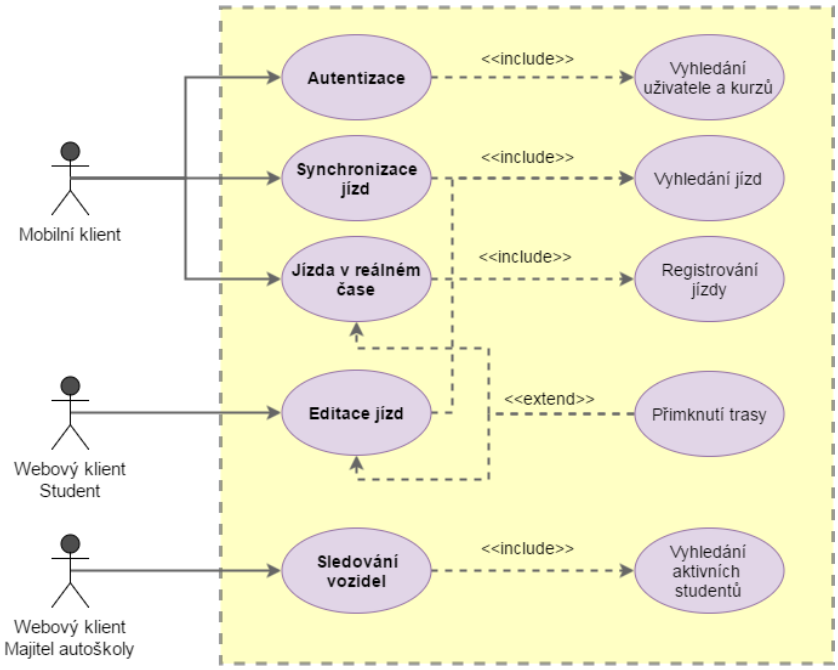
komunikace zapouzdřují. V případě delegování zpráv mezi aktivitou a službou je nutné *IPC* implementovat pomocí *AIDL* (kapitola 6.2).

V momentě spuštění on-line záznamu tras pomocí tlačítka v uživatelském prostředí, je nejprve nutné spustit službu. K tomu slouží právě *IPC*. V rámci služby bude implementováno naslouchání aktuální pozice s využitím GPS. `ContentResolver` při změně lokace požádá pomocí *Uri* `ContentProvider` o uložení. V případě úspěchu upozorní všechny registrované `ContentObservery`, které pozorují na konkrétní *Uri*. Z důvodu zákazu jakýchkoliv náročných operací v UI vlákně, je úkolem pozorovatele obsahu vytvořit nové vlákno pro vykreslení aktuálních dat.

V případě povolení on-line záznamu tras, bude přímo ve službě implementováno spojení se serverem podle kapitoly 7, které umožní odesílání souřadnic geobodů v reálném čase. Webový klient následně může zažádat server o automatické zasílání pozice konkrétního žáka nebo žáků dané autoškoly.

8.3 Návrh struktury serverové aplikace

Na obrázku 8.3 je uveden jednoduchý návrh případu užití navrhované serverové aplikace, ve které figurují účastníci *mobilní klient*, *student* a *majitel autoškoly*, kde poslední dva přistupují k serveru pomocí webového prohlížeče a mobilní klient využívá chytrý telefon s platformou *Android*.



Obrázek 8.3: Návrh struktury serverové aplikace.

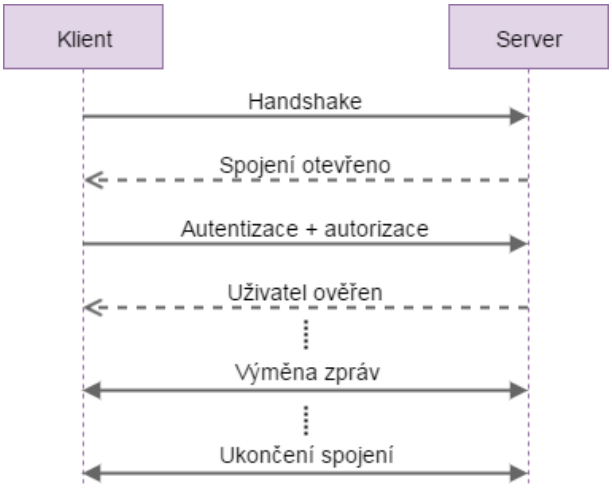
Mobilní klient, jak je uvedeno v případu užití na obrázku 8.3, využívá serverovou aplikaci k autentizaci uživatele, kdy zároveň dojde ke stažení aktivních kurzů podle databáze *MySQL* specifikované v kapitole 2.4. Po přihlášení studenta autoškoly v aplikaci chytrého telefonu jsou pro něj zpřístupněny případy užití pro synchronizování jízd se serverem a vytvoření jízdy pro sledování vozidla v reálném čase.

Účastník *Student* reprezentuje přihlášeného žáka autoškoly na portálu *DoAutoškoly.cz*, kterému je umožněno procházet své naměřené trasy v rámci kurzu a následně je upravovat. Každá upravená trasa uživatelem se bude automaticky editovat podle reálné vozovky na mapě.

Aktér *Majitel autoškoly* reprezentuje, jak již název napovídá, přihlášeného majitele autoškoly na webových stránkách *DoAutoškoly.cz*, který komunikuje a interaguje se serverem za účelem získání informace o právě aktivních studentech vykonávající svoji jízdu. U těchto žáků je možné sledovat aktuální polohu vozidla v reálném čase.

Autentizace klienta

Všichni aktéři budou k serveru přistupovat pod shodnou IP adresou a portem. Proto je nutné jednotlivé účastníky odlišit již při autentizaci k serverové aplikaci, aby přistupovali pouze k té části aplikace, na kterou mají právo, a nedošlo k neoprávněnému odcizení dat. Konkrétní návrh komunikace mezi jednotlivými klienty a serverem je znázorněn na obrázku 8.4.

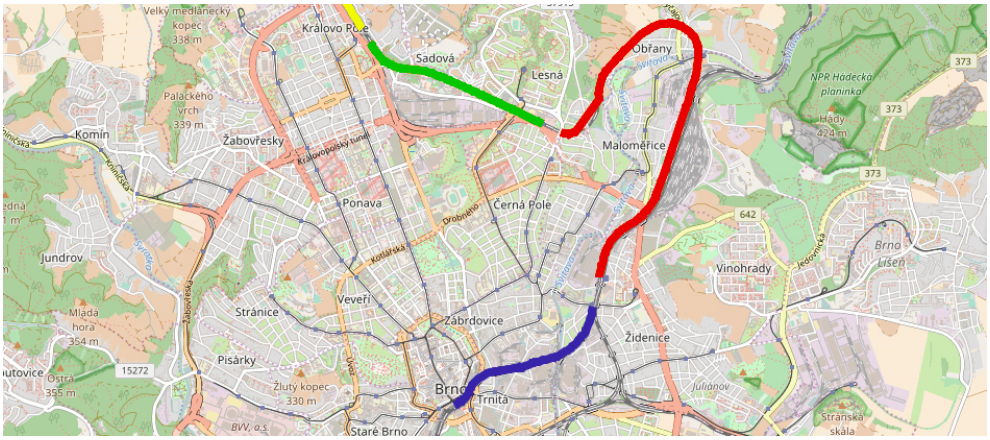


Obrázek 8.4: Návrh komunikace mezi serverem a klientem.

Po vytvoření spojení mezi účastníky komunikace podle kapitoly 7 je nutné emitovat zprávu o typu uživatele společně s přihlašovacími údaji. Po autentizaci a autorizaci je v dalším kroku spuštěno oboustranné delegování informací podle případu užití specifikovaného podle schématu na obrázku 8.3.

8.4 Formát tras

GPS v mobilních zařízeních je levného charakteru a je tak velmi citlivé na okolní vjemy při potřebě získat přesnou polohu zařízení. Při snímání trasy v reálném čase jsou vlivem špatných podmínek běžné výpadky signálu satelitů. V takovém případě by docházelo k nereálnému vykreslení trasy, protože vzdálenost dvou sousedících geografických bodů může být příliš velká. Z tohoto důvodu budou jednotlivé trasy rozděleny na segmenty souvisejících geobodů. Konkrétní příklad lze vidět na obrázku 8.5. S každou delší ztrátou signálu je tak nutné vytvořit nový segment, kde každý segment musí obsahovat minimálně jeden geobod a každá trasa musí obsahovat minimálně jeden segment.



Obrázek 8.5: Segmenty trasy na mapovém podkladu.

8.5 Serializace tras

Pro výměnu zpráv mezi serverem a aplikací je výhodné využít formát *JSON*, protože *MongoDB* používá pro ukládání právě *JSON* dokumenty.

Zprávy mezi serverem a klientem budou nejčastěji obsahovat geografické informace o aktuální poloze nebo dokonce celé zaznamenané trasy uživatele. Pro tyto účely by bylo vhodné využít *GeoJSON*[3], formát pro prostorová data, který umožňuje i zápis dalších neprostorových atributů. Nakonec bylo nutné od tohoto standardu upustit, a to hned z několika důvodů.

Souřadnice v *GeoJSON* jsou podle *RFC 7946*[4] zapsány v poli, kde první hodnotou atributu `coordinates` je zeměpisná délka a druhou zeměpisná šířka. Většina rozhraní tuto konvenci respektuje, ale například *MongoDB* udává souřadnice přesně v opačném pořadí.

Vzhledem ke konkrétnímu užití v této aplikaci, *GeoJSON* obsahuje redundantní atributy a zanoření dat. Při přenosu tras mezi serverem a klientem je nutné také uchovávat například přesnost, čas a nadmořskou výšku jednotlivých geobodů. Tyto údaje by bylo nutné podle specifikace *GeoJSON* dále zanořovat a z pohledu návrhu databáze *MongoDB* a *SQLite* by vznikaly další komplikace. Z těchto důvodů je ideální navrhnout vlastní formát, který se bude více svou strukturou blížit oběma databázím.

Na obrázku 8.6 je uveden konkrétní návrh přenosu zpráv ve formátu *JSON*. `TRACK` reprezentuje jednu konkrétní naměřenou trasu uživatele (`uid`) pomocí chytrého telefonu. Parametr `_id` definuje globální identifikátor jízdy, který vzniká při uložení jízdy.

Objekt obsahuje identifikaci studenta (`sid`) včetně zvolené autoškoly (`aid`) a skupiny kurzu (`gid`) podle existující databáze webu *DoAutoškoly.cz* definované v kapitole 2.4. Objekt trasy dále obsahuje v pořadí čas začátku a konce jízdy, čas poslední editace a čas smazání. Zatímco atribut `segments` obsahuje vždy originální naměřené segmenty (pole objektu `SEGMENT`), tak atribut `segmentsEdit` reprezentuje upravené segmenty uživatelem nebo po automatickém přimknutí trasy k reálné vozovce (pole objektu `SEGMENT`). Objekt `SEGMENT` obsahuje pole objektu `POINT` a analogicky s objektem pro jízdu čas začátku a konce segmentu. Objekt `POINT` reprezentuje (v pořadí atributů) čas získání souřadnice, zeměpisná délka, zeměpisná šířka, poskytovatel (GPS nebo internet), nadmořská výška geobodu, přesnost souřadnice, rychlost objektu v daném bodě a azimut.

[3] http://geojson.org
[4] https://tools.ietf.org/html/rfc7946

```
TRACK = {
  _id: String,
  sid: Number,
  uid: Number,
  gid: Number,
  aid: Number,
  timeStart: Number,
  timeEnd: Number,
  timeLastUpdate: Number,
  timeDelete: Number,
  segments: [SEGMENT],
  segmentsEdit: [SEGMENT]
}
```

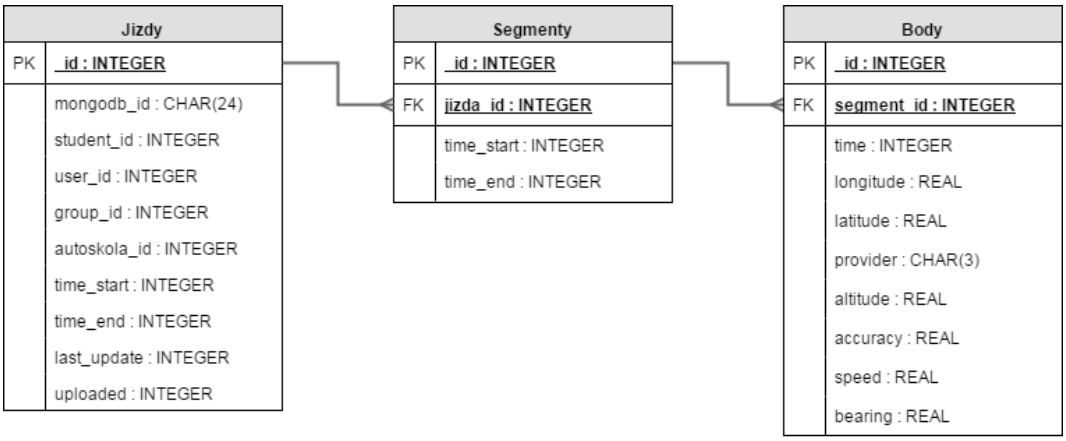
```
SEGMENT = {
  timeStart: Number,
  timeEnd: Number,
  points: [POINT]
}
```

```
POINT = {
  time: Number,
  lon: Number,
  lat: Number,
  provider: String,
  altitude: Number,
  accuracy: Number,
  speed: Number,
  bearing: Number
}
```

Obrázek 8.6: Návrh formátu tras v *JSON* pro přenos zpráv.

8.6 Návrh databáze SQLite

Návrh *SQLite* databáze pro mobilní zařízení s operačním systémem *Android*, zobrazený na obrázku 8.7, se skládá ze tří tabulek a zohledňuje segmentování tras, které bylo blíže uvedeno v kapitole 8.4.



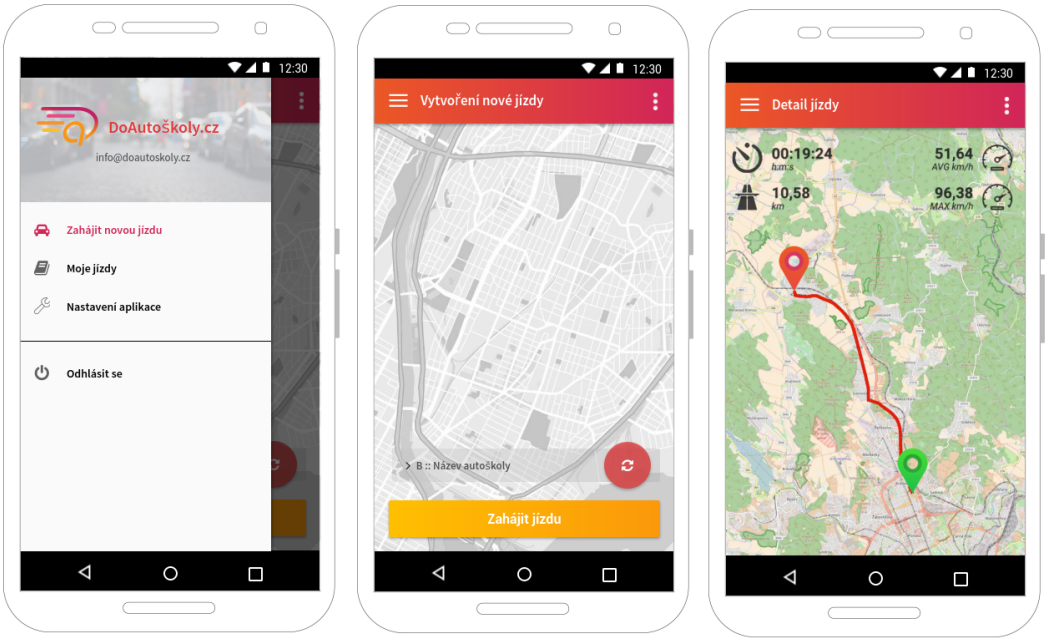
Obrázek 8.7: Schéma navrhované databáze *SQLite* pro mobilní zařízení.

Podobně jako u formátu pro přenos zpráv mezi serverem a klientem z kapitoly 8.5, tabulka `Jizdy` zastupuje záznamy o všech jízdách v rámci jednoho chytrého telefonu. Zde musíme uvažovat situaci, kdy jeden uživatel vlastní a používá více zařízení pro záznam tras, které následně chce synchronizovat se serverem. Proto z důvodu konzistence dat je nutné v mobilním zařízení uchovávat identifikátor záznamu databáze *MongoDB* po nahrání dat na server. Tabulka s jízdami obsahuje atomické údaje pro identifikaci studenta, uživatele, autoškoly, skupinu kurzu a dále časové údaje o konkrétním záznamu. Samozřejmě opět každá trasa obsahuje minimálně jeden segment, který má své atributy značící zahájení a ukončení. Každý segment zahrnuje alespoň jeden geobod s atomickými prostorovými a neprostorovými informacemi. Pomocí množiny zeměpisných souřadnic, včetně času pořízení, lze zpětně sestrojit a vykreslit kompletní trasu.

8.7 Návrh uživatelského rozhraní mobilní aplikace

V návaznosti na podkapitolu 8.2, kde byl vysvětlen návrh struktury aplikace pro platformu *Android*, a podkapitolu 8.3 vysvětlující případ užití pro tento typ uživatele, bude v této části textu popsán koncept uživatelského rozhraní vyobrazený na obrázku 8.8.

Aplikace je koncipována od samého začátku z pohledu uživatelského prostředí velmi jednoduše. Design byl navrhnut podle stávajícího webového portálu *DoAutoškoly.cz*. Sestává se z jedné hlavní aktivity obsahující tzv. `Navigation drawer` neboli navigaci, která vlastní reference na fragmenty reprezentující vytvoření nové jízdy, seznam ukončených jízd studenta s návazností na detail jízdy a v neposlední řadě nastavení aplikace. Navigace s položkami je navrhnuta na prvním obrázku zleva. Aktivita dále vlastní prvek `Action bar` značící fragment zobrazený na popředí a ikonu pro otevření navigace.



Obrázek 8.8: Návrh základních fragmentů mobilní aplikace platformy *Android*.

8.8 Návrh uživatelského rozhraní webové aplikace

Webová aplikace má primárně sloužit z pohledu studentů k editaci naměřených jízd a z pohledu majitelů autoškol ke sledování aktivních vozidel. Na rozdíl od mobilní a serverové aplikace, které je nutné od základu navrhnout a realizovat, webové rozhraní bude implementováno formou modulu a bude využita stávající implementace layoutu portálu *DoAutoškoly.cz* včetně veškerých kaskádových stylů. V době psaní této diplomové práce je obsahová část webu rozdělena do dvou sloupců, kde levá a zároveň větší část bude využita pro mapový náhled, a naopak pravá menší část bude sloužit k seznamu jízd, respektive k aktuálnímu seznamu přihlášených studentů absolvujících svoji jízdu.

Kapitola 9

Použité jazyky a technologie

Před samotnou implementací projektu podle návrhu z předchozí kapitoly 8 je velmi důležité rozhodnutí, pomocí jakých technologií, implementačních jazyků a frameworků bude práce realizována. Při volbě implementačních možností je nutné mít na paměti, že jednotlivé části systému budou mezi sebou komunikovat v reálném čase. Tím pádem je nutné nalézt takovou množinu možností a technologií, která mezi sebou nebude v rozporu. Tato kapitola je rozdělena na čtyři stěžejní části, které lze implementovat různými technologiemi. Konkrétně se jedná o serverovou aplikaci, webovou aplikaci, mobilní aplikaci, a nakonec společná část týkající se komunikace mezi nimi. Dále zde budou popsány technologie pro realizaci editování uložených jízd.

9.1 Serverová aplikace – Node.js a TypeScript

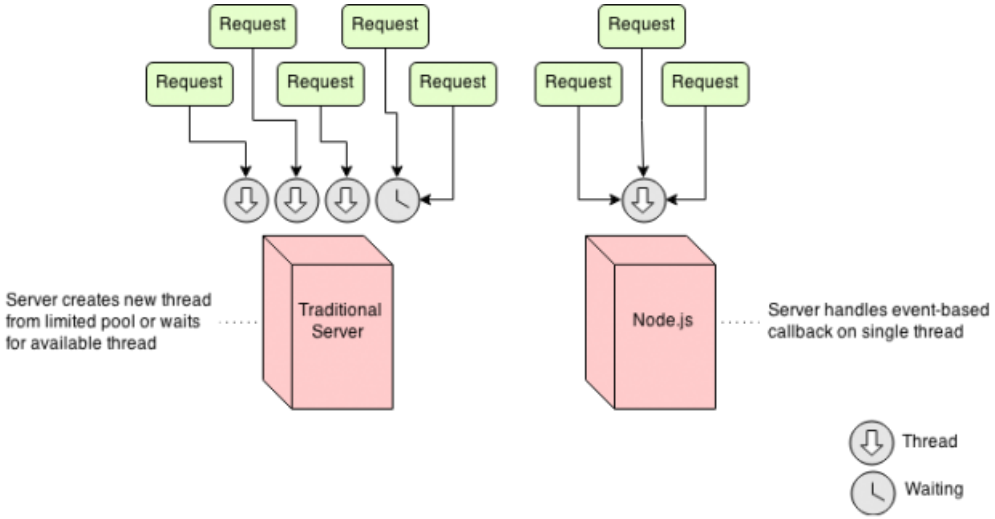
Pro vývoj serverové části byla zvolena dnes velmi rozšířená a vysoce výkonná platforma *Node.js*[1] umožňující server-side využití programovacího jazyka *JavaScript*, kde aplikace mohou být spouštěny v běhovém prostředí na operačních systémech Linux, Windows a OS X.

Základním principem aplikačního prostředí *Node.js* je, že vytvořené aplikace jsou v běhovém prostředí (tzn. runtime) neustále spuštěné a zpracovávají veškeré požadavky v jednom vlákně. Na rozdíl od *HTTP* serveru *Apache*, který pro každý požadavek vytváří nový speciální proces. Tento hlavní rozdíl mezi těmito servery je znázorněný na obrázku 9.1.

Cílem *Node.js* je maximalizovat výkon a minimalizovat latenci. To je docíleno tím, že veškeré aplikace realizované nad *Node.js* jsou řízené událostmi, vlastní jediné vlákno a používají neblokující asynchronní zpracování vstupu a výstupu. [14]

Vzhledem k tomu, že *Node.js* je postavený na programovacím jazyku *JavaScript*, potřebuje nějaký interpret, který se postará o vykonání našeho skriptu. Toto běhové prostředí je postaveno na moderním *engine V8*, který je původně vytvořený společností *Google* pro webový prohlížeč *Chrome*. Základ tohoto interpreta je napsán v jazyce *C++*. Nad touto knihovnou existuje ještě jedna vrstva implementovaná v jazyce *C*, která zajišťuje například zpracování příchozích událostí. Interpret je dále rozšířený o funkce, které umožňují vývojářem vytvořeným skriptům přistupovat k souborům nebo síťovým funkcím. Díky tomu lze vytvořit aplikaci, která naslouchá na daném portu a slouží tak jako základ pro vytvoření skutečné *real-time* aplikace. Podle webových stránek *Node.js* se dokonce jedná o silnou stránku tohoto aplikačního prostředí. [4]

[1] https://nodejs.org



Obrázek 9.1: Porovnání schéma zpracování požadavků u *Apache* a *Node.js*. [14]

Základem serverových implementací v *JavaScriptu* je mimo jiné *CommonJS*, jehož cílem je sjednotit základní postupy. Standard *CommonJS* definuje dva nástroje. Jedním je funkce `require` umožňující importovat konkrétní modul do aktuálního prostoru a druhým je naopak objekt `module`, který slouží k exportování ze jmenného prostoru.

Node.js nabízí dále databázi různých balíčků (modulů) pomocí *Node Package Manager* (dále jen *npm*), které lze libovolně procházet na webových stránkách *npmjs.com*[2] a následně vybrané instalovat. Instalace balíčků se provádí příkazem `npm install`, který je již součástí instalace *Node.js*. Všechny nainstalované balíčky vývojářem jsou uvedeny v souboru `package.json`, který zaručuje rychlou instalaci a aktualizaci na novější verze.

Jedním z balíčků, který lze stáhnou a nainstalovat je *TypeScript*[3]. Jedná se o *open-source* programovací jazyk postavený nad jazykem *JavaScript*, který přináší zejména statické typování. Každý vývojář si jistě umí představit situaci, kdy vyvíjí síťovou aplikaci, kde přenos zpráv probíhá ve formátu *JSON*, ale v čistém *JavaScriptu* nezná schéma příchozích dat. V tomto směru *TypeScript* přináší usnadnění vývoje rozsáhlých aplikací, které jsou založené na *JavaScriptu*.

Jedinou drobnou nevýhodou je, že kromě balíčků *npm* musíme znát i potřebné soubory s deklaracemi a rozhraními. Tento problém řeší balíček *typings*, který přináší deklarace většiny knihoven. Jejich seznam je veden ve veřejném repozitáři[4]. Instalace balíčků se provádí obdobně jako u *npm* příkazem `typings install`.

Pro vývoj serverové a klientské části, implementované pomocí *TypeScriptu* a dalších technologií zmíněných v této kapitole, byl vybrán nekomerční a *open-source* softwarový nástroj *Atom.io*[5]. Tento textový editor je k dispozici pro platformy *Linux*, *Windows* a *OS X*. Díky bohaté škále pluginů *Atom* nabízí podporu pro vývoj programů v jazycích *C/C++*, *PHP*, *CoffeScript* a mnoho dalších. K nim patří i jazyk *TypeScript*, kterému nechybí ani zvýraznění syntaxe, našeptávač, automatický překlad při uložení souboru,

[2] https://www.npmjs.com/
[3] https://www.typescriptlang.org/
[4] https://github.com/DefinitelyTyped/DefinitelyTyped
[5] https://atom.io/

TODO list a mnoho dalších nástrojů usnadňující vývoj. Editoru nechybí ani podpora frameworků *Node.js*, *React.js* a dalších.

9.2 Webová aplikace – TypeScript a React.js

Jako hlavní implementační technologie pro vývoj webových stránek je značkovací jazyk *HTML*. Jak již bylo zmíněno v kapitole 8 zabývající se návrhem, stěžejní klientské části budou dodány formou modulu do stávajícího webového portálu *DoAutoškoly.cz*, kde byla použita nejnovější verze značkovacího jazyku. Konkrétně se jedná o specifikaci *HTML 5*[6], jejíž nejnovější verze se datuje k říjnu 2014, která byla vydána po hlavičkou konsorcia *W3C*.

Jako dynamický jazyk na straně klienta se používá skriptovací jazyk *JavaScript* ve standardizované verzi *ECMAScript*, který se stále může pyšnit titulem nejpoužívanější jazyk tohoto typu. [33]

Podobně jako u serverové aplikace je využita platforma *Node.js* společně s již známým programovacím jazykem *TypeScript*. Jedná se o velkou sílu tohoto aplikačního rozhraní. Osobně považuji za velkou výhodu možnost implementovat serverovou a klientskou aplikaci pomocí stejných technologií. Jediný rozdíl spočívá v tom, že výsledná aplikace neběží jako u serveru v běhovém prostředí, ale je nutné vygenerovat výsledný skript formou modulu, který lze snadno importovat do existujícího *HTML* kódu libovolného webového portálu.

Řešením je balíček *webpack*[7] stažitelný pomocí *npm*. Jedná se o velmi mocný nástroj, který slouží k sbalení všech modulů do jednoho souboru za účelem použití *JavaScriptu* u klienta v prohlížeči. Základním stavebním kamenem jsou tzv. *loadery*. Každý formát (.js, .ts, .tsx, .png, ...) může mít svůj vlastní. Díky *webpack* lze sbalit prakticky cokoliv do jednoho souboru včetně obrázků, kaskádových stylů apod.

Zatím šlo pouze o princip vytvoření samotného JS kódu, který je interpretován až po stažení celé webové stránky u uživatele v libovolném prohlížeči. Pro zjednodušení a urychlení práce je dále vhodné použití nějakého JS frameworku, které obecně umožňují elegantnější a kvalitnější návrh aplikace. Pro tento projekt byla zvolena knihovna pro vytváření uživatelských rozhraní *React.js*, který je rovněž formou modulu stažitelný přímo z *npm*.

Výhodou *React.js* založeného na jazyku *JavaScript* je, že programátory nenutí se učit novou syntaxi, ale naopak kód syntakticky vypadá jako klasické *HTML*. *React.js* se snaží celý layout svým návrhem dělit na komponenty, kde každá její třída obsahuje jedinou povinnou metodu `render` obsahující vždy jediný kořenový element objektového modelu dokumentu (DOM). Každá komponenta lze v aplikaci využít nespočetně krát. Asi největší výhodou tohoto frameworku je jeho rychlost. Aplikace často obsahují i několik stovek DOM elementů. V případě vykreslování celého DOMu při jakékoliv změně může být značně pomalé. Proto *React.js* využívá tzv. *Diffing Algorithm*, kdy nejprve porovná dva stromy elementů a následně vykresluje pouze nevyhnutelné změny. [6] Konkrétně u webového modulu, definovaného návrhem v kapitole 8, bude rychlost vykreslování důležitým faktorem. Vzhledem k tomu, že se jedná o aplikaci, která bude zejména sledovat vozidla v reálném čase, bude nezbytné často renderovat seznam aktivních uživatelů, aktuální rychlost vozidla apod.

[6] https://www.w3.org/TR/html5/
[7] https://webpack.js.org/

9.3 Mobilní aplikace – Java

Při výběru implementačních možností pro vývoj mobilní aplikace platformy *Android* v době psaní této diplomové práce není moc na výběr. Lépe řečeno, je pouze jedna elegantní možnost. Tou je programovací jazyk *Java* a *Android Studio*[8], oficiální vývojové prostředí pro *Android* vyvinuté společnostmi *Google* a *JetBrains*. Ještě donedávna, podobně jako v publikaci [3], bylo doporučované prostředí *Eclipse*, které bylo velmi oblíbené, ale dle mého názoru pomalu ale jistě ustupuje ze scény. Kromě rychlosti samotného prostředí, je stále složitější instalace a podpora různých knihoven pro nejmodernější verze platformy *Android*. S vývojem *Android Studia* přestaly být k dispozici některé důležité knihovny ve formátu *.jar*, a to zejména z důvodu zavedení nástroje *Gradle*[9], jehož instalace v prostředí *Eclipse* není vždy bezproblémová a kompatibilní.

Gradle je především nástroj pro automatizaci sestavování programu postavený na programovacím jazyku *Groovy*. Konkrétně se jedná o DSL (Domain Specific Language), který nám umožňuje popsat chtěnou zautomatizovanou událost. Konkrétně v *Android Studiu* se používá především k definování parametrů pro překlad a také k určení výčtu knihoven, které jsou automaticky staženy nebo, pokud vývojář požaduje, aktualizovány z konkrétních serverů.

Součástí instalace *Android Studia* je *Android SDK*[10] (Software Development Kit). Jedná se o kompletní sadu vývojových, ladících a testovacích nástrojů dostupné na operačních systémech Linux, Windows nebo OS X. Součástí jsou, mimo jiné, i emulátory zařízení, na kterém běží konkrétní verze operačního systému *Android* a slouží primárně k testování vyvíjených aplikací. Většina balíčků je volně dostupná ke stažení pomocí *SDK Managera*, který je součástí instalace *Android Studia*.

S vývojem aplikace *Android* se pojí i přesně definovaná adresářová struktura projektu určující přesný typ obsahu: [3]

- **AndroidManifest.xml** – soubor popisující sestavenou aplikaci včetně veškerých použitých oprávnění a komponent,
- **aidl/** – adresář uchovávající definované programové rozhraní, viz *AIDL* v kapitole 6.2,
- **java/** – adresář uchovávající zdrojový kód v jazyce *Java*,
- **res/** – adresář uchovávající „*suroviny*“, jako jsou například ikony, uživatelské rozhraní ve formátu *XML* apod., které jsou součástí zkompilovaného kódu:
  - **res/drawable/** – obsahuje vektorové a rastrové obrázky nebo definované elementy ve formátu *XML*,
  - **res/layout/** – obsahuje definované uživatelské rozhraní ve formátu *XML*,
  - **res/menu/** – obsahuje definici kontextových nabídek ve formátu *XML*,
  - **res/mipmap/** – obsahuje ikonu aplikace, která je definovaná v souboru *AndroidManifest.xml*,
  - **res/values/** – obsahuje soubory s hodnotami ve formátu *XML*, kde mezi nejdůležitější patří:

[8] https://developer.android.com/studio/index.html
[9] https://gradle.org/
[10] https://developer.android.com/studio/releases/sdk-tools.html

    - **res/values/colors.xml** – uchovává externalizované definice barev,
    - **res/values/styles.xml** – uchovává externalizované definice stylů určitých prvků uvedených v adresáři `res/layout/`,
    - **res/values/dimens.xml** – uchovává externalizované definice dimenzí,
    - **res/values/strings.xml** – uchovává externalizované definice řetězců
  - **res/xml/** – obsahuje ostatní soubory ve formátu *XML*, konkrétně se například jedná o definici uživatelského nastavení aplikace `PreferenceScreen`.

Výhodou této strukturalizace je především snadná rozšiřitelnost lokalizace nebo zavedení jiného layoutu pro různé velikosti obrazovek. Adresáře `res/**/` mohou mít v názvu sufix znamenající fakt, že mají být použity v konkrétním případě.

9.4 Komunikace – Socket.IO

Socket.IO[11] je open-source framework umožňující obousměrnou komunikaci v reálném čase založenou na událostech. Tento rámec pracuje na všech platformách, zařízeních a webových prohlížečích a rozhodně mu nechybí ani rychlost a spolehlivost[12].

Framework se skládá ze dvou částí, z klienta a serveru. Serverová část je postavena na řešení *HTTP* serveru *Node.js* v balíčku *express*. *Socket.IO* není rozhodně pouhá implementace *WebSocket*, ale naopak podporuje několik typů transportních metod. Velkou výhodou je fakt, že vývojář se příliš nemusí zajímat o rozdíly mezi různými transportními způsoby. Zde uvedu pouze metody *WebSocket*, *Long polling* a *Polling* (podobně jako v kapitole 7, která je vysvětluje) z toho důvodu, že u platformy *Android* pomocí tohoto frameworku nelze jiné použít. Při implementaci může vývojář jednoduše pomocí množiny určit transportní metody, které se mohou použít. V případě, že prohlížeč první z nich nepodporuje, přejde na další.

Jak už bylo naznačeno výše, implementace je k dispozici v několika jazycích. Balíček *socket.io* je volně stažitelný pro *JavaScript* přímo v *npm*. Oficiální implementace, jenž mají velmi podobné aplikační rozhraní, jsou k dispozici i pro jazyky *Java*, *C++* a *Swift*. Pro ostatní prostředí existují od tvůrců třetích stran.

Základní rysy frameworku *Socket.IO* jsou:

- **Spolehlivost** – spojení je vytvořeno i v přítomnosti proxy, firewall nebo antivirového programu,
- **Podpora znovupřipojení** – rámec, pokud není nastaveno jinak, automaticky připojuje odpojeného klienta od serveru,
- **Detekce odpojení** – rámec automaticky detekuje na straně serveru i klienta neočekávané odpojení účastníka spojení pomocí určení časového limitu, který je sdělen při vytváření spojení,
- **Rooms** – zprávy mohou být rozesílány skupině uživatelů v jednom čase pomocí tzv. *místností* definovaných vývojářem, do kterých se jednotliví klienti připojují,
- **Multiplexing** – aby bylo možné oddělit jednotlivé zájmy aplikace, rámec umožňuje vytvořit tzv. *Namespaces*, které vytvoří separátní komunikační kanály, ale ve skutečnosti sdílejí stejné připojení.

[11] https://socket.io/
[12] https://github.com/socketio/socket.io

9.5 Automatická úprava geodat – OSRM

Open Source Routing Machine[13] (dále jen *OSRM*) je projekt nabízející řešení, mimo jiné, pro hledání nejkratší cesty v síti komunikací na mapě nebo pro upravení trasy podle existující cesty.

Knihovna *OSRM backend* je implementovaná v jazyce *C++*, která nabízí rozhraní pomocí protokolu *HTTP*. To zaručuje interakci z jakéhokoliv jiného implementačního jazyka. Z tohoto důvodu musí být knihovna spuštěna ve svém vlastním běhovém prostředí. Bohužel neexistuje žádný balíček pro spuštění (kromě operačního systému *Windows*, pro který jsou připravené soubory *.exe* ke stažení[14]), ale naopak je nutné stáhnout *open-source* zdrojový kód a následně jej zkompilovat podle uvedeného návodu v oficiální dokumentaci[15] projektu. Výsledkem kompilace jsou binární soubory `osrm-extract`, `osrm-contract` a `osrm-routed`. [23]

Veškeré výpočty se provádí z předem připravených souborů, které jsou vytvořeny z *Open Street Map* ve formátu *PBF* stažitelné ze serveru *geofabrik*[16]. Nejprve je nutné pomocí programu `osrm-extract` extrahovat data z komprimovaného formátu *PBF* do formátu *OSRM*. Následně se provede spuštěním programu `osrm-contract` přepočítání tras pro běh serveru. [23]

Spuštěním serveru binárním souborem `osrm-routed` jsou následně přes rozhraní[17] k dispozici tyto služby:

- **Nearest** – nalezne nejbližší bod na síti komunikací k zadané souřadnici,
- **Route** – nalezne nejrychlejší cestu mezi dvěma souřadnicemi,
- **Table** – vypočítá trvání nejrychlejší cesty pro každou dvojici souřadnic na trase,
- **Match** – přimkne množinu souřadnic k reálné síti komunikací,
- **Trip** – řeší známý *Problém obchodního cestujícího*[18].

Toto řešení spočívá ve spuštění druhé aplikace v běhovém prostředí vedle serverové aplikace navržené v kapitole 8, kde tyto dvě části budou mezi sebou komunikovat pomocí protokolu *HTTP*.

9.6 Manuální úprava geodat – Leaflet

Manuální úprava naměřených jízd studentem bude probíhat pouze prostřednictvím přihlášeného webového klienta k serverové aplikaci. Leaflet[19] je *open-source* knihovna implementovaná v jazyce *JavaScript* umožňující interaktivní zobrazení map v dokumentech *HTML*, včetně dotykových zařízeních. Knihovna nabízí bohaté aplikační rozhraní s množstvím doplňujících *pluginů* nevyjímaje podpory pro zobrazování a editování vektorových dat.

[13] https://github.com/Project-OSRM/osrm-backend
[14] https://github.com/Project-OSRM/osrm-backend/wiki/Windows-Compilation
[15] https://github.com/Project-OSRM/osrm-backend/wiki/Building-OSRM
[16] http://download.geofabrik.de/europe/
[17] https://github.com/Project-OSRM/osrm-backend/blob/master/docs/http.md
[18] Problém obchodního cestujícího: úkolem je najít nejkratší možnou trasu, která prochází všemi zadanými souřadnicemi a končí ve stejné souřadnici jako začala.
[19] http://leafletjs.com/

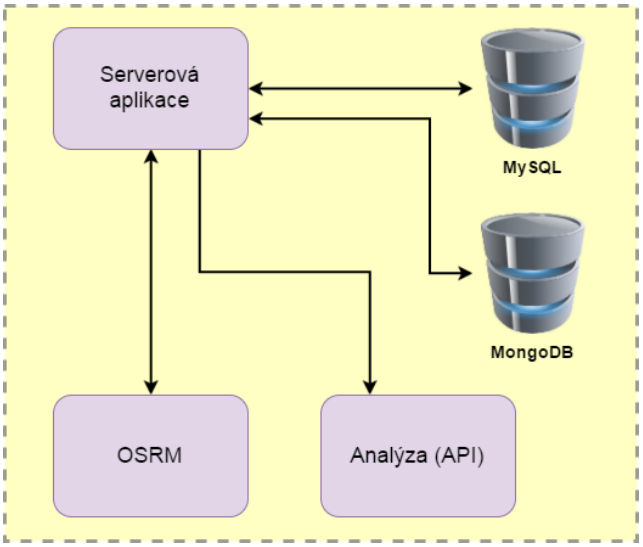
Kapitola 10

Implementace

V této kapitole diplomové práce bude čtenář seznámen se samotnou implementací celého systému. Celá implementace se prakticky dělí na tři části, konkrétně na realizaci mobilní aplikace pro platformu *Android*, zhotovení modulu pro stávající webový portál *DoAutoškoly.cz* a nakonec nejdůležitější část v podobě serverové aplikace, která zaštiťuje komunikaci mezi klienty a nabízí rozhraní pro manipulaci s datovou vrstvou. Prioritním cílem této kapitoly je čtenáři nastínit řešení a možné problémy, které mohou vzniknout například pří získávání geografických dat pomocí mobilního zařízení nebo při komunikaci v reálném čase mezi webovým klientem a mobilním zařízením. V závěru kapitoly je kladen důraz na bezpečnost komunikace mezi serverovou aplikací a klienty. Jedná se o část práce, kterou nelze u správného návrhu opominout, protože zejména při sledování vozidel v reálném čase se jedná o velmi citlivé údaje.

10.1 Architektura serverové aplikace

Struktura serverové aplikace byla blíže popsána v návrhu v kapitole 8.1 včetně případu užití v kapitole 8.3. Vzhledem k výběru technologií pro implementaci celého systému, konkrétně editování uložených jízd, bylo nutné doplnit návrh struktury serverové aplikace o další část.



Obrázek 10.1: Schéma serverové aplikace.

Na obrázku 10.1 je vyobrazený návrh, který je doplněn o další serverovou aplikaci podle kapitoly 9.5 umožňující upravení posloupnosti geografických bodů podle mapového podkladu (dále jen OSRM).

Jak již bylo zmíněno v předešlých kapitolách, jedná se o serverovou aplikaci, která je přístupná pomocí *IP* adresy a portu z libovolného webového prohlížeče nebo z aplikace v chytrém telefonu. Serverová aplikace stále slouží primárně ke zpracování veškerých příchozích požadavků od klientů, která využívá dle návrhu dvě databáze MySQL a MongoDB, dále využívá aplikační rozhraní pro oznámení nových dat v databázi MongoDB kvůli analýze a nově aplikaci OSRM, se kterou serverová komunikace dle kapitoly 9.5 komunikuje výhradně pomocí protokolu *HTTP* a obě části (Serverová aplikace a OSRM) běží ve svém vlastním běhovém prostředí.

V následující části textu této kapitoly budou podchyceny nejdůležitější části realizace serverové aplikace jako celku včetně OSRM, databází a využití aplikačního rozhraní.

10.1.1 Struktura a nastavení aplikace

Veškeré potřebné soubory pro běh serverové aplikace jsou obsaženy na přiloženém disku. Soubory `package.json`, `tsconfig.json` a `typings.json` obsahují definice potřebných knihoven a pokyny pro transpilaci implementačního jazyka *TypeScript* do interpretujícího jazyka *JavaScript*, viz. kapitola 9.1. Podrobný návod pro překlad a spuštění serverové aplikace je obsažen v souboru `README.md`.

Z důvodu bezpečnosti citlivých dat, zdrojové soubory uvedené na disku neobsahují žádné přihlašovací údaje k externím databázím apod. Složka `src` uchovává veškeré zdrojové soubory ve formátu *.ts*, konkrétně se jedná o tyto části:

- **DatabaseMongo/** – Uchovává modul pro práci s databází *MongoDB*, který je implementovaný v rámci již zmíněné diplomové práce Bc. Martina Šouláka. Obsahuje důležitý soubor `src/config/ConfigDB.ts` exportující rozhraní s nastavením pro připojení k databázi.
- **DatabaseMySQL/** – Uchovává třídy pro práci s databází *MySQL*, včetně souboru `MySQLSettings.ts` exportující rozhraní s nastavením pro připojení k databázi.
- **ServerApps/** – Uchovává třídy obsluhující vstupní požadavky klientů, viz kapitola 10.1.4.
- **Utils/** – Uchovává pomocné třídy týkající se kryptografie, logování stavu aplikace, čas apod.
- **Server.ts** – Hlavní soubor spouštějící hlavní třídu `ServerMain`.
- **ServerMain.ts** – Třída definující vytvoření serverové aplikace včetně naslouchání na daném portu.
- **ServerHandler.ts** – Třída obsluhující příchozí události serverové aplikace, viz kapitola 10.1.4.
- **ServerSettings.ts** – Soubor obsahující rozhraní s nastavením serverové aplikace. Definuje zejména port, na kterém bude aplikace naslouchat příchozí události, a také použití *OSRM*, které není pro běh samotné aplikace prioritní, ale spíše pomáhá zlepšit naměřené trasy a jízdu v reálném čase.

Výše definované pouze poukazuje na základní strukturu zdrojového kódu serverové aplikace. Následující podkapitoly vysvětlují mimo jiné jednotlivé části struktury, zejména ty pro zpracování vstupních událostí. Bližší informace k jednotlivým souborům jsou podány v podobě komentářů přímo ve zdrojovém kódu.

10.1.2 Databázová vrstva

Specifikací návrhu řešení struktury systému v kapitole 8.1 byly stanoveny dvě databáze, konkrétně *MongoDB* a *MySQL*.

V případě databáze *MongoDB* pro ukládání naměřených jízd studentů bylo dodáno aplikační rozhraní formou modulu, který byl implementován v jazyce *TypeScript*. Jak bylo zmíněno v návrhu řešení práce, modul byl vyvíjen v rámci diplomové práce Bc. Martina Šouláka. Pro správné spuštění modulu je důležité jej nahrát do adresáře serverové aplikace. Celý balíček je následně přeložen společně s celým programem z *TypeScriptu* do interpretovatelného jazyka *JavaScript*. Do aplikačního rozhraní vstupují a vystupují objekty reprezentující jednotlivé jízdy, segmenty a množinu geografických bodů. Rozhraní objektů je totožné s návrhem uvedeným v kapitole 8.5. Rozhraní dále implementuje metody pro vkládání naměřených jízd, jejich získávání, upravování a také další metody pro ukládání jízd v reálném čase.

Databáze slouží pro použití v této práci výhradně pro autentizaci a autorizaci uživatelů. Vzhledem k rozsahu používání této databáze, nebyl použit žádný ORM framework, který by zajistil mapování objektů na relační databází a naopak. K přístupu k databázi byl využit modul *mysql*[1] volně stažitelný z *npm*.

Obě databáze v systému jsou přístupné pouze lokálně a není tak možné získávat údaje například přímo z mobilního zařízení nebo webového klienta. Klienti musí veškeré informace získávat prostřednictvím serverové aplikace.

Třídy reprezentující databáze *MongoDB* a *MySQL* jsou koncipovány podle návrhového vzoru *Singleton*. Jedináček zaručuje jedno jediné připojení ke konkrétní databázi v rámci celé serverové aplikace, které je neustále aktivní, a sdílení jediné instance mezi různými části programu bez nutnosti předávání pomocí parametrů.

10.1.3 Vytvoření serverové aplikace

Jako hlavní implementační jazyk byl pro serverovou aplikaci po konzultaci vybrán *TypeScript*, který je následně přeložen na *JavaScript* obsahující hlavní třídu `ServerMain`. Vytvořením instance této třídy pomocí souboru *Server.js* ve složce *build* je spuštěna serverová aplikace. Pro spuštění samotného programu je využíván modul *supervisor*[2], který při každé změně jakéhokoliv souboru nebo při neodchycené chybě restartuje aplikaci. Tím je zaručeno neustálé spuštění serverové aplikace v běhovém prostředí, vyjímaje případu, kdy z jakékoliv příčiny přestane pracovat celý server, na kterém je umístěn program.

Alogoritmus 10.1 demonstruje vytvoření naslouchání serverové aplikace na konkrétním portu. Nejprve jsou pomocí funkce `require`, která je součástí *CommonJS*, zavedeny moduly *Express*, *Http* a *SocketIO*. Jednotlivé moduly zpřístupňují exportované funkce, které umožňují sestavení serveru. Pomocí modulu *Express*[3] a *Node.js* je vytvořena aplikace založená na událostech a asynchronním zpracováním. Dále modul *Http* exportuje

[1] https://www.npmjs.com/package/mysql
[2] https://github.com/petruisfan/node-supervisor
[3] http://expressjs.com/

funkci `createServer` s jedním parametrem pro definování obslužné funkce pro příchozí spojení, kterou zaštiťuje právě rámec *Express*. Funkce `createServer` vrací objekt `Http.Server` předaný frameworku *Socket.IO*. Nakonec stačí již vytvořit samotné naslouchání na konkrétním portu, který je předaný parametrem funkci `listen` patřící objektu `Http.Server`. Metoda `listen` dále vlastní jako druhý nepovinný parametr *IP* adresu, na niž server naslouchá, a jako třetí parametr *callback* funkci, která je volána ve chvíli navázání serveru na zadaný port. V tuto chvíli je rámec *Socket.IO* svázán s *HTTP* serverem a budou mu delegovány vnější příchozí zprávy od klientů. Rámec *Socket.IO* vytváří nad použitým protokolem určitou míru abstrakce a zároveň poskytuje velmi podobnou funkcionalitu webovým prohlížečům i aplikacím chytrého telefonu. Konkrétní zpracování těchto událostí bude pečlivě popsáno v následující podkapitole 10.1.4.

```
// Inicializace modulů
var app    : Express.Express = Express();
var server : Http.Server     = Http.createServer(app);
var io     : SocketIO.Server = SocketIO.listen(server);

// Naslouchání serveru na daném PORTU
server.listen(PORT, function(){
   this.runServer(io);
});
```

Algoritmus 10.1: Vytvoření naslouchání serverové aplikace.

10.1.4 Zpracování vstupních událostí

Komunikace mezi serverovou aplikaci a klientskou stranou je implementována pomocí knihovny *Socket.IO*, u které je dle kapitoly 7 a 9.4 využíván prioritně protokol *WebSocket*. V případě, že strana klienta nepodporuje protokol *WebSocket*, je nahrazen jinou komunikační metodou. Z pohledu vývojáře serverové aplikace zajišťuje *Socket.IO* takovou abstrakci, že princip zpracování příchozích požadavků je stále stejný.

V momentě otevření komunikace ze strany klienta, ať už z webového prohlížeče nebo chytrého telefonu, je otevřen nový *socket* - programová schránka obsahující informace o propojených objektech. Pomocí *socketu* lze delegovat zprávy z klienta na server, ale i opačně ze serveru ke klientovi. Rámec *Socket.IO* dokonce umožňuje zasílat zprávy skupině připojených klientů.

Z pohledu vývojáře obsahuje příchozí zpráva dvě části. První je textový řetězec uvádějící název, který může být libovolný a volí jej programátor. Jediná podmínka zní, že název zprávy musí být známý druhé straně. Druhý parametr je *callback* funkce, která je volána při nové příchozí události a obsahuje v parametrech libovolné objekty včetně další *callback* funkce reprezentující tzv. *acknowledge* – potvrzující zpráva, která navíc může obsahovat další data.

Podobně i odchozí zpráva vlastní v prvním parametru textový řetězec, který značí název zprávy. Další parametry obsahují přenášená data v podobě libovolného objektu nebo *callback* funkci, ve které je očekávaná odpověď od druhé komunikační strany. Jedinou podmínkou pro komunikaci je, aby odchozí zpráva měla shodné (očekávané) parametry jako u odchycení příchozích událostí.

Algoritmus 10.2 nastiňuje využití a princip komunikace mezi klientem a serverem, respektive ukázka odeslání, přijmutí a potvrzení zprávy pomocí frameworku *Socket.IO*.

```
// Klient - připojení k serverové aplikaci
var socket=IO.connect("http://127.0.0.1:3001", <ConnectOpts>{});
socket.emit("název zprávy", data, function(result){
   // zpracování odpovědi
});

// Server - připojen nový klient
this.io.sockets.on("connection", function(socket : Socket){
   // Zpracování příchozí události
   socket.on("název zprávy", function(data, acknowledge){
      // Odpověď na událost
      acknowledge(data);
   });
});
```

Algoritmus 10.2: Připojení uživatele k serverové aplikaci.

Jak už víme, komunikace začíná na straně klienta. Voláním funkce `connect` frameworku *Socket.IO* je provedeno připojení k serveru na určité *IP* adrese a portu definované prvním parametrem. Druhý parametr definuje nastavení připojení, konkrétně přesná definice transportních metod a jejích priorita, timeout apod. V případě úspěchu je vrácen objekt *socketu*. Pro přesnou kontrolu stavu klienta jsou zde k dispozici předem definované události, konkrétně `connect`, `disconnect`, `connect_error` apod.

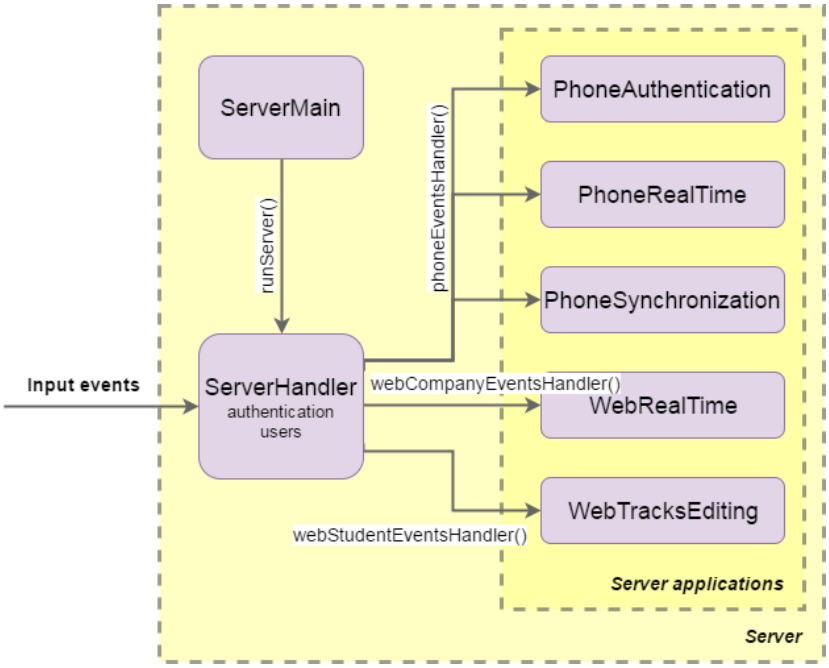
Odchycení události `connection` pomocí funkce `on` definuje nového příchozího klienta na straně serveru, kde *callback* funkce obsahuje parametr typu `Socket` obsahující, mimo jiné, identifikátor *socketu*. V této chvíli se jedná o neautentizovaného uživatele. Principu ověření bude věnována vlastní podkapitola 10.1.5.

Komunikace je i nadále událostmi řízená. Každá akce na straně serveru i klienta může být doprovozena vlastní definovanou událostí, na kterou patřičná strana může odpovědět. Rámec *Socket.IO* dokonce umožňuje definovat stejnou událost vícekrát, což je například výhodné při odpojení klienta na straně serveru, kdy je zapotřebí provést více akcí současně. Události lze také různě třídit do konkrétních tříd podle autentizovaného a následně autorizovaného klienta, jak tomu naznačuje schéma na obrázku 10.2.

Jak už bylo zmíněno v kapitole 10.1.3, třída `ServerMain` spustí naslouchání událostí a následně vytvoří instanci třídy `ServerHandler`. Tato instance je zodpovědná za odchycení vstupních událostí od připojených klientů z webové a mobilní aplikace. Každý uživatel je bezprostředně po připojení k serverové aplikaci ověřen pomocí příchozího autentizačního objektu, viz kapitola 10.1.5. Podle návrhu struktury serverové aplikace v kapitole 8.3 se mohou v systému objevit tři aktéři, jsou jimi uživatel mobilního zařízení platformy *Android*, student přihlašující se pomocí webového klienta a v neposlední řadě majitel autoškoly, který je taktéž autentizován webovým prohlížečem.

Po úspěšném přihlášení daného aktéra, je jeho objekt zařazen do seznamu klientů daného typu, viz kapitola 10.1.7, a také je jeho *socket* předán konkrétní obslužné třídě, kde je definováno zpracování vstupních událostí pro konkrétní typ uživatele:

- **PhoneAuthentication** – definuje typy událostí pro autentizaci uživatele chytrého telefonu včetně informací o aktuálních kurzech v přihlášených autoškolách,



Obrázek 10.2: Schéma zpracování vstupních událostí.

- **PhoneSynchronization** – definuje typy událostí pro stažení a nahrání nových nebo upravených jízd studenta,
- **PhoneRealTime** – definuje typy událostí pro vytvoření a ukončení nové jízdy a také událost pro příchozí souřadnice pří snímání jízdy v reálném čase,
- **WebRealTime** – definuje typy událostí pro odeslání aktivních studentů dané autoškoly a také odesílání souřadnic studenta v reálném čase,
- **WebTracksEditing** – definuje typy událostí pro stažení seznamu jízd, získání konkrétní jízdy podle určeného identifikátoru a uložení upravené jízdy studenta.

10.1.5 Autentizace klientů

Autentizace jednotlivých klientů k serverové aplikaci probíhá podle návrhu z kapitoly 8.3 pomocí zaslání zprávy okamžitě po připojení. Další často používanou metodou je zaslání přihlašovacích údajů v tzv. *connection stringu* zároveň s připojením k serveru, kde je nevýhodná právě viditelnost údajů v *URL* a s tím související bezpečnost, které bude věnována kapitola 10.5.

Autentizace klientů na serveru je funkčně rozdělena do dvou kategorií, konkrétně se skládá z klientů užívající webový prohlížeč a klientů užívající mobilní telefon. U chytrého telefonu je situace jednoduchá, protože pro ověření je možné odeslat zašifrovaný objekt obsahující e-mailovou adresu a zašifrované heslo. V tomto případě nelze pro heslo použít hashovací funkci, protože současný portál *DoAutoškoly.cz* uchovává hesla pomocí symetrické blokové šifry *Bcrypt*, která používá generovanou kryptografickou sůl, a tudíž nelze použít obyčejné porovnání řetězců. Z tohoto důvodu je nutné serverové aplikaci doručit heslo v původní podobě zadané uživatelem s využitím šifry.

Při ověření webového klienta je situace poměrně komplikovanější, protože implementační jazyk *JavaScript* se interpretuje až na straně klienta. Vytvoření zašifrovaného objektu by tak mohl kdokoliv následně rozluštit přímo z kódu a tím získat přístup k velmi citlivým údajům. *JavaScript* by ze stejného důvodu neměl ani znát přesné přihlašovací údaje. Řešením je při interpretaci *PHP* kódu na straně serveru vytvořit opět objekt obsahující identifikátory přihlášeného uživatele, který se po zašifrování předá *JavaScriptu*, tzv. *token-based*. Poté u klienta nejsou viditelné žádné údaje, pouze šifra v podobně textového řetězce.

Serverová aplikace pomocí tajného klíče rozšifruje příchozí řetězec, sestaví objekt a v případě úspěchu dále ověří klienta, žádajícího o citlivé údaje, pomocí databáze *MySQL*, která je přístupná pouze lokálně na serveru.

10.1.6 Správa klientů

Pro každý typ uživatele existuje globální pole uchovávající seznam přihlášených uživatelů k serverové aplikaci. Veškeré uživatele spravuje třída `Users`.

V případě webových klientů není problém povolit vícenásobné přihlášení uživatele pod shodnými přihlašovacími údaji. Je běžné, že uživatel si webovou aplikaci otevře na více zařízeních nebo záložkách a tím pádem dojde i k nové autentizaci uživatele na serveru. Indexování uživatelů v poli je tady zajištěno pomocí identifikátoru příchozího *socketu*.

Naopak v případě klientů užívající chytré telefony je potřeba k této problematice přistoupit důsledněji. Je třeba vzít do úvahy, že tito uživatelé budou dávat k dispozici aktuální polohu vozidla, ve kterém vykonávají svoji jízdu. Sice je nepravděpodobné, že by student vlastnil v době jízdy více mobilních telefonů s datovým připojením, ale i na tuto situaci je nutné brát zřetel. Je tedy nutné eliminovat vícenásobné přihlášení klienta, které by znamenalo nesoulad v určení aktuální polohy. Podobně by docházelo ke komplikacím při synchronizaci nových jízd. Jako index seznamu klientů je zvolen identifikátor přihlášeného uživatele podle databáze *MySQL*. V případě další žádosti o autentizaci stejného klienta je odeslána odpověď s chybovým stavem znamenající vícenásobné přihlášení k serverové aplikaci.

10.1.7 Správa místností

Při implementaci sledování vozidel v reálném čase je důležité určit cílovou skupinu uživatelů, kteří z pohledu serveru mají právo na získání aktuální polohy, a naopak nedošlo k nežádoucímu získání citlivých dat. Při sledování vozidla bude mobilní zařízení odesílat svoji polohu směrem k serveru bod po bodu, kde tyto body budou dále rozesílány jednomu nebo více klientům (majitelům autoškoly) užívající webového klienta. Komunikace bude mezi klienty probíhat pouze jednosměrně.

Pro usnadnění této komunikace je využit framework *Socket.IO*, který umožňuje definovat tzv. místnosti a následně rozesílat zprávy všem uživatelům v jedné místnosti zároveň. Bezprostředně po autentizaci majitele autoškoly je *socket* s využitím funkce `join` přidán do místnosti podle identifikátoru autoškoly. V tuto chvíli je možné delegovat zprávu, například s novou pozicí, všem přítomným uživatelům s využitím funkce `io.sockets.in("název místnosti").emit("jméno zprávy", <Object>{})`. Podobným způsobem jsou delegovány zprávy o příchozím a odchozím studentovi, který žádá o měření jízdy v reálném čase resp. o ukončení stávající jízdy. V momentě ukončení sledování vozidel autoškoly je klient odebrán z místnosti funkcí `leave`.

10.1.8 Logování stavu aplikace

Při běhu serverové aplikace dochází k výpisu zpráv na standardní a chybový výstup, který zaštiťuje třída `Logger`, jejíž implementace se nachází v adresáři `Utils`.

Zaznamenávány jsou veškeré důležité události, které aplikace vykoná, včetně zachycených signálů. Jedním z nich je i zachycení neodchycené výjimky, která by znamenala pád celé serverové aplikace, a tím by došlo k omezení ostatních uživatelů.

```
------------------------------------------------------------------
---------------------- HELLO, I'M SERVER -----------------------
------------------------------------------------------------------
[2017-04-22 20:19:10.399 > ServerMain]              INFO        MONGO CREATE                        Connection created successfully.
[2017-04-22 20:19:10.705 > DoAuto_MySQL]            INFO        MYSQL CREATE                        Pripojeno k databazi MySQL. ID: 2
------------------------------------------------------------------
--------------------------- RUNNING ----------------------------
------------------------------------------------------------------
[2017-04-22 20:19:10.745 > ServerMain]              INFO        CREATE SERVER                       Hostname: Kuba-NB, Port: 3002, PID: 4112
[2017-04-22 20:19:10.746 > ServerHandler]           INFO        SERVER LISTENING                    Server naslouchá klientum ...
[2017-04-22 20:19:36.291 > DoAuto_MySQL]            DEBUG       MYSQL PING                          Pripojeni je stale aktivni
[2017-04-22 20:19:37.706 > ServerHandler]           DEBUG       WEB STUDENT AUTHENTICATED - '1029'  ID: 1029, Name: null
[2017-04-22 20:19:37.707 > ServerHandler]           DEBUG       WEB STUDENT - '1029' - REQUEST      Zadost o editovani jizd studenta
[2017-04-22 20:19:37.712 > WebTracksEditing]        DEBUG       WEB STUDENT - '1029' - RECEIVE      pozadavek na stazeni jizd studenta
[2017-04-22 20:19:37.992 > WebTracksEditing]        DEBUG       WEB STUDENT - '1029' - ANSWER       odeslano 24 jizd
[2017-04-22 20:19:44.351 > WebTracksEditing]        DEBUG       WEB STUDENT - '1029' - RECEIVE      pozadavek na stazeni jizdy studenta podle id
[2017-04-22 20:19:44.617 > WebTracksEditing]        DEBUG       WEB STUDENT - '1029' - ANSWER       Poslani jizdy 58dcf3b3917dee0f76de7368
```

Obrázek 10.3: Výpis běžící serverové aplikace.

Události serverové aplikace vystupují celkem v pěti kategoriích, konkrétně (v pořadí podle váhy hlášení) `VERBOSE`, `DEBUG`, `INFO`, `WARNING` a `ERROR`. Zatímco první dvě kategorie slouží spíše pro účely testování, tak `INFO` slouží primárně pro definování stavu, ve kterém se serverová aplikace nachází, jak je například vidět na obrázku 10.3. Poslední dvě kategorie reprezentují chybový stav, kde `WARNING` definuje odchycené chyby, ale zároveň se nejedná o standardní stav programu, a `ERROR` definuje neodchycené chyby, u kterých je nutné vykonat nápravu. Každá zpráva dále obsahuje přesný čas, třídu výskytu hlášení, *tag* určující skupinu nebo přihlášeného uživatele, a nakonec samotnou zprávu definující důvod výskytu.

10.2 Architektura mobilní aplikace

Návrh uživatelského rozhraní společně s *real-time* komunikací, databází a serializací byl specifikován v kapitole 8.2. Kapitola 9.3 naopak definovala implementační možnosti a nástroje. V této kapitole bude čtenář seznámen s prvky aplikace pro platformu *Android*, se kterými byl docílen již zmíněný návrh. Dále zde bude nastíněn princip implementace snímání a synchronizace jízd.

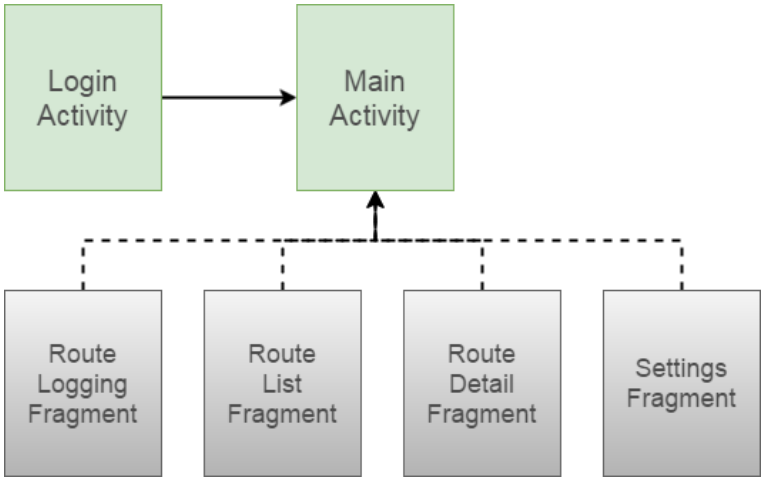
Před implementací jakékoliv aplikace pro platformu *Android* je nutné v souboru `AndroidManifest.xml` a `build.gradle` definovat minimální verzi *API*, která bude podporována. V tomto konkrétním případě *API 15*, které vyplývá z kapitoly 6.1 a odpovídá použitým prvkům pro uživatelské prostředí.

Veškeré zdrojové soubory implementovaných tříd a rozhraních se nachází v balíčku (*package*) `cz.doautoskoly.mojejizdy`.

10.2.1 Uživatelské rozhraní

Uživatelské rozhraní aplikace se sestává ze dvou aktivit, které jsou potomkem třídy `AppCompatActivity` a slouží tak jako základ pro zobrazení prostředí interakce s uživateli chytrého telefonu.

První aktivitou, která je uživateli spuštěna na popředí obrazovky bezprostředně po nainstalování aplikace, je `LoginActivity`. Grafické rozvržení `activity_login.xml` definuje



Obrázek 10.4: Zjednodušené schéma implementace aplikace *Android*.

prvky `EditText` a `Button` umožňující přihlášení studenta. Po úspěšném přihlášení uživatele, které bude blíže popsáno v kapitole 10.2.2, je na popředí obrazovky otevřena aktivita `MainActivity`, která se sestává z prvků definující `Navigation drawer` a `Action bar`.

Grafické rozvržení `activity_main.xml` uchovává kořenový *layout* `DrawerLayout`, který slouží k vytvoření posuvného menu ze strany displeje. *Layout* definuje dva dceřiné elementy:

- **NavigationView** – je vizuální interaktivní prvek, který vykreslí menu s definovanými položkami v souboru `activity_main_drawer.xml` a hlavičkou (podle návrhu z kapitoly 8.7) definovanou v souboru `activity_main_header.xml`,
- **CoordinatorLayout** – je vizuální interaktivní prvek definovaný v souboru `activity_main_frame`, který se skládá z dalších dvou hlavních prvků:
  - **AppBarLayout** – je vizuální interaktivní prvek, který slouží k vytvoření kontejneru na animaci hlavičky obsahující titulek aktivního fragmentu a ikonu pro otevření menu,
  - **FrameLayout** – je grafické rozvržení, které se používá pro uchování jednoho prvku (v tomto případě pro uchování jednoho fragmentu).

Hlavní aktivita, která je zobrazena pouze autentizovanému uživateli, se sestává ze čtyř fragmentů, viz obrázek 10.4. Jednotlivé fragmenty se po kliknutí vkládají do grafického rozvržení `FrameLayout`. Činnost jednotlivých fragmentů bude dále popsána v následujících kapitolách.

10.2.2 Autentizace uživatele

Autentizace z pohledu serverové aplikace a následného ověření byla popsána v kapitole 10.1.5. Tato kapitola nastíní problematiku z pohledu uchování citlivých informací v platformě *Android*.

Po úspěšném vyplnění přihlašovacích údajů uživatelem a potvrzení formuláře, aktivita `LoginActivity` spustí nové vlákno vytvořením nové instance třídy odvozené od `AsyncTask`, která zajišťuje plnou kontrolu nad prováděnou operací na pozadí

a možnost reakce uživatelského rozhraní na různé podněty. Samotnou autentizaci zapouzdřuje třída `Authentication` (balíček `SocketIO`), která implementuje abstraktní třídu `ASocketIO` sloužící k vytvoření spojení mezi klientem a serverem včetně emitování zprávy `phone_authentication` podle kapitoly 10.1.5 zabývající se autentizací klientů z pohledu serverové aplikace. V případě úspěchu serverové aplikace vrátí informace o uživateli a otevřených kurzech. Následně je otevřena hlavní aktivita `MainActivity`.

Informace o přihlášeném uživateli je nutné uchovávat po celou dobu existence aplikace v mobilním zařízení nebo do okamžiku ručního odhlášení. Pro ukládání dat tohoto typu v platformě *Android* existuje tzv. *Shared Preferences*, ke kterým mohou přistupovat všechny aktivity aplikace. Naopak jsou chráněny proti přístupu z ostatních aplikací. Veškeré údaje se ukládají principem *klíč-hodnota*.

Objekt *Shared Preferences* je získán metodou `Context.getSharedPreferences(String name, int mode)`. Tento objekt dále zapouzdřuje, mimo jiné, metodu `edit` vracející objekt třídy `SharedPreferences.Editor`, která slouží pro ukládání dat.

Jak bylo zmíněno v kapitole 10.1.5, heslo uživatele je z důvodu ověření nutné delegovat serveru v originální textové podobě. To znamená, že i heslo musí být v telefonu podobným způsobem uchováno, ale to zároveň nese bezpečnostní riziko. Z tohoto bezpečnostního důvodu jsou veškeré informace ukládány až po zašifrování algoritmem *AES*, který užívá pro šifrování i dešifrování shodný klíč.

10.2.3 Snímání jízdy

Snímání jízdy pomocí chytrého telefonu se řídí návrhem z kapitoly 8.2. Samotné získávání souřadnic s využitím aplikačního rozhraní *GPS* obstarává registrovaná služba aplikace. Konkrétně třída `LocationService` v balíčku `Location`. Naopak uživatelské rozhraní zaštiťuje fragment `RouteLoggingFragment`, jehož součástí jsou ovládací prvky pro snímání jízdy a případné statistiky při běžící službě.

Po stisknutí tlačítka snímání jízdy, jsou veškeré body ukládány do databáze *SQLite*. Oproti návrhu z kapitoly 8.6 byl do tabulky *Body* po konzultaci přidán atribut `isFiltered`, jehož důvod bude blíže popsán v další části této kapitoly. Pro řádné ukončení jízdy je nutné splnit podmínku neprázdnosti. Konkrétně počet bodů musí být větší, než počet segmentů s tím, že prázdné segmenty nejsou ukládány.

Tato kapitola se dále bude zabývat principem získání, uložení a prezentování získaných dat pomocí aplikačního rozhraní *GPS*. Komunikace v reálném čase je zde záměrně opominuta a bude ji věnována vlastní kapitola 10.3.

Komunikace se službou

Jak už víme z kapitoly 8.2 a 6.2, aktivita s fragmentem spouštějící snímání jízdy pracuje v odlišném vlákně než služba, která obhospodařuje získávání dat.

Služba je spuštěna metodou `Context.startService(Intent)` po stisknutí tlačítka na úrovni fragmentu. Veškerou komunikaci mezi službou a aktivitou nadále zaštiťuje třída `LocationManager` v balíčku `Location`. Metoda `startNewServiceConnection` vytvoří nové spojení se službou, které je zajištěno vykonáním metody `Context.bindService(Intent, ServiceConnection, int flag)`. Na straně služby metoda `bind` vrací instanci třídy `ServiceRemoteAidl` (potomek abstraktní třídy `IServiceRemote.Stub`) implementující chování rozhraní *AIDL*. Po vytvoření spojení třída `LocationManager` získá tzv. *binder* tohoto rozhraní. V tento moment může fragment využívat rozhraní a vzdáleně tak přistupovat k aktuálním informacím služby jako je například poslední souřadnice, segment

a statistické údaje. Pomocí rozhraní se zároveň určuje chování služby z pohledu uživatele, tedy zapnutí/vypnutí trasování, pauza/pokračování a zapnutí/vypnutí tzv. *listenerů*.

Nastavení aplikačního rozhraní

K získávání zeměpisných souřadnic je využíváno aplikační rozhraní platformy *Android*, které bylo popsáno v teoretické části práce (kapitola 6.5). Metodou `requestLocationUpdates` je registrován *listener* pro konkrétního poskytovatele. Po konzultaci a nastudování metod z kapitoly 6.6 bylo rozhodnuto využít pro lokalizaci pouze *GPS*. Důvodem byla vysoká nepřesnost a rozptyl souřadnic, které byly získány pomocí datového připojení.

Parametr	**Hodnota**	**Jednotka**
Časová frekvence	1000	milisekundy
Minimální změna pozice	0	metry

Tabulka 10.1: Nastavení aplikačního rozhraní pro získání pozice.

Již zmíněná metoda zpracovává další dva parametry, které jsou definovány pomocí tabulky 10.1. Časová frekvence byla stanovena na jednu sekundu. Vzhledem k tomu, že studenti, vyjímaje prvních jízd, jezdí převážně ve městě, jejich maximální rychlost se bude přibližovat 50 km/h. Vzhledem k nezkušenosti budoucích řidičů spíše k 40 km/h. Při dodržení této rychlosti získáme souřadnici zhruba každých 10 metrů. Při klesání rychlosti by byly jednotlivé souřadnice zbytečně příliš blízko u sebe. Z tohoto důvodu se volí druhý parametr, u kterého byla prvotní myšlenka omezit získávání souřadnic na minimální hranici dvou metrů (polovina orientační délky osobního automobilu), aby došlo k eliminaci redundantních bodů, například při stání na křižovatce. Tento parametr byl nakonec po konzultaci s Bc. Martinem Šoulákem nastaven na hodnotu 0 m, na minimální vzdálenost tedy nebude brán zřetel. Je to z důvodu zavedení analýzy zabývající se plynulostí jízdy, která lze vypočítat podle shluku bodů.

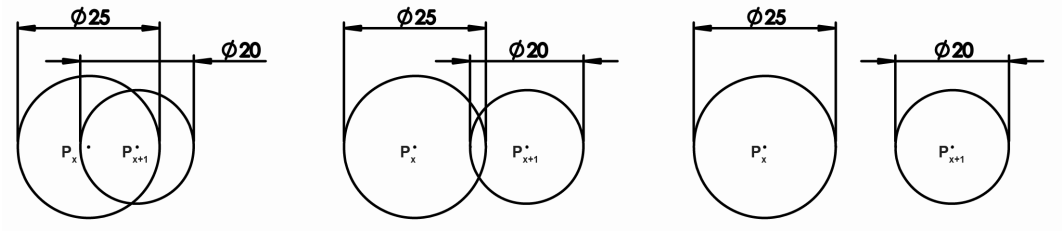
Filtrování bodů

Po registraci *listenerů*, metoda `onLocationChanged` vrací polohu s různou přesností v závislosti na počtu viditelných satelitů a okolním prostředí. Filtrování těchto geobodů bylo rozděleno do dvou fází:

- **Primární** – Primární filtrování kontroluje pouze přesnost získané souřadnice pomocí *API*. Přesnost, kterou si můžeme představit jako kružnici kolem souřadnice, je stanovená na 20 m. Při testování bylo zjištěno, že při optimálních podmínkách mobilní zařízení získá přesnost 3 až 10 m, naopak standardní přesnost se pohybuje zhruba v rozmezí 10 až 18 m. I z důvodu následné automatické úpravy tras nebyla povolená horší přesnost. Geobody jsou ukládány do interní databáze telefonu pouze v případě, že splní tuto podmínku.
- **Sekundární** – Sekundární filtrování bylo zavedeno zejména ve snaze eliminovat redundanci geobodů při stání vozidla na křižovatce nebo při parkování a kvůli automatickému přimknutí trasy podle reálné vozovky. Veškeré souřadnice po sekundárním filtrování jsou ukládány do databáze s `boolean` parametrem `isFiltered`. V sekundárním filtrování byly zavedeny tyto metody:

– vzdálenost mezi bodem P_x a P_{x+1} je větší než součet jejich přesností, viz obrázek 10.5, kde pouze třetí případ splňuje tuto podmínku,

– rychlost mezi bodem P_x a P_{x+1} nesmí přesáhnout 250 km/h nebo klesnout pod 5 km/h.



Obrázek 10.5: Filtrování geobodů při snímání jízdy.

Vykreslení jízdy

Za vykreslování snímané jízdy je zodpovědný fragment `RouteLoggingFragment`, který před jízdou inicializuje *controller* pro ovládání mapy a také registruje *observery* pro odchycení nové pozice nebo segmentu v databázi. Veškeré třídy a rozhraní pro vykreslení mapy se nachází v balíčku `View.Map`.

Třída `MapController` implementuje zmíněný *controller* sloužící pro vykreslení tras, segmentů a přidání bodu k již vytvořeným segmentům. Instanci této třídy je nutné v parametru konstruktoru předat rozhraní `IMapView`, které definuje metody pro ovládání a vykreslení elementárních prvků na mapě předem určeného poskytovatele vývojářem. Rozhraní zaručuje budoucí snadnou vyměnitelnost mapových zdrojů. Třída `MapViewOSM`, implementující rozhraní `IMapView`, zpřístupňuje mapové podklady *Open Street Maps* a funkcionalitu balíčku *Osmdroid*[4].

Fragment dále registruje dva *observery* pomocí metody `registerContentObserver`, kterou zapouzdřuje `ContentResolver`. Konkrétně se jedná o třídy `ObserverSegment` a `ObserverPoint` (potomci abstraktní třídy `ContentObserver`) definující jedinou *override* metodu `onChange`, která reaguje na změnu podle předem definované *Content Uri* 10.1, resp. 10.2 (kde $\#_1$ je identifikátor jízdy a $\#_2$ je identifikátor segmentu).

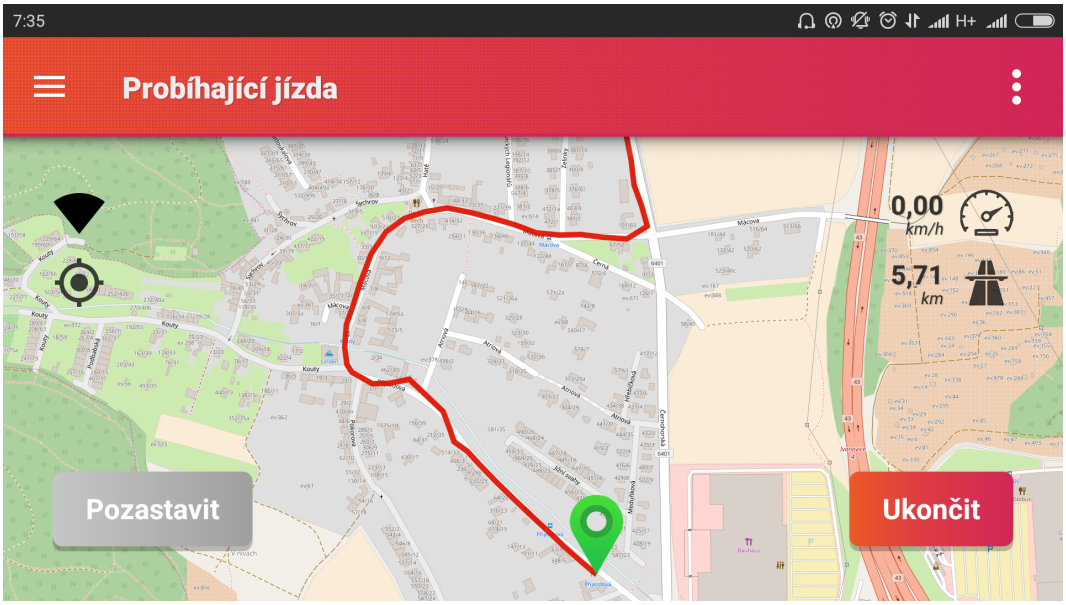
$$content : //cz.doautoskoly.mojejizdy.provider/jizdy/\#_1/segmenty \tag{10.1}$$

$$content : //cz.doautoskoly.mojejizdy.provider/jizdy/\#_1/segmenty/\#_2/geobody \tag{10.2}$$

V okamžiku, kdy služba získá novou souřadnici, požádá `ContentProvider` pomocí *Content Uri* o uložení, který následně upozorní všechny *observery* pozorující na dané *Uri*. `ObserverPoint` získá novou souřadnici, kterou dále deleguje instanci třídy `MapController` a dojde k překreslení *polyline*.

V momentě vytvoření nového segmentu, dojde podobným principem k informování *observeru*, který je registrován s *Uri* 10.1. Kromě delegování zprávy instanci třídy `MapController` informující o nutnosti vytvoření nového segmentu, je nutné odhlásit

[4] https://github.com/osmdroid/osmdroid



Obrázek 10.6: Vykreslení snímání jízdy.

původní *observer* `ObserverPoint` a registrovat jej s novou *Uri* 10.2 obsahující aktuální identifikátor segmentu.

Metoda `onChange` je pouze informována o změně dat na konkrétní *Uri* bez znalosti posledního identifikátoru. Konkrétní instance třídy `GeoPoint` je získána pomocí *AIDL* přímo z běžící služby. Naopak instanci třídy `Segment` je nutné získat přímo z databáze pomocí `ContentResolver`. A to z toho důvodu, že může dojít k získání nové souřadnice dříve, než je *observer* informován o novém segmentu. V takovém případě se načtou předchozí záznamy z databáze zároveň s informací o segmentu.

Dosud byl popsán princip vykreslování dat, které jsou získány službou a následně jsou uloženy do interní databáze. Na obrázku 10.6 znázorňujícím snímání jízdy je zřetelné, že je zároveň nutné vykreslit informace, které nejsou ukládány a jejich existence závisí pouze na službě. Konkrétně se jedná o signál *GPS*, informaci o sledování nebo další průběžné statistiky o jízdě. Pro tyto účely byla implementována třída `HeartBeat`, která je potomkem abstraktní třídy `TimerTask` a definuje jedinou *override* metodu `run`. Metodou `Timer.schedule(TimerTask, int, int)` je nastaveno občerstvování uživatelského rozhraní s počátečním zpožděním a opakováním 1 s (stejné jako frekvence zisku nových souřadnic).

Možné problémy

Spuštěnému trasování je nutné zaručit neustálý běh do okamžiku vypnutí jízdy uživatelem. Prostředí pro aplikace platformy *Android* je stavěné spíše pro krátkodobé operace, ať už z pohledu uživatele nebo z pohledu využitých prostředků systému. Vzhledem k těmto faktům je nutné se vyvarovat několika situacím.

Vypnutí snímání jízdy musí být legální. Spuštěná služba má sice přidělen svůj vlastní proces, ale z hlediska systému je stále vázaná s aktivitou, která ji spustila. Zde může dojít k problému v okamžiku, kdy uživatel, ať už nechtěně nebo cíleně, odebere aplikaci ze seznamu právě spuštěných aktivit. V tomto momentě je nutné aktivitu násilným

způsobem znovu spustit s fragmentem nové jízdy, v opačném případě by uživatel mohl přijít o naměřená data. Přímo ve službě k tomu slouží *override* metoda `onTaskRemoved`, která zachytí odebrání služby z paměti. U některých verzí platformy *Android* (podle výrobce) lze tato funkce zakázat.

Podobná situace může nastat v případě neočekávaného vypnutí zařízení, například z důvodu nedostatku baterie nebo se může jednat i o akci uživatele. Z tohoto důvodu je registrován *Broadcast receiver* s názvem `BootReceiver`, který aplikaci spustí po naběhnutí systému v případě, že nebyla řádně ukončena jízda. Ale opět se jedná o funkci, která u některých verzích lze zakázat vlastníkem zařízení.

Vážným problémem, který může nastat, je zprvu užitečná funkce telefonu kontrolující využívání procesoru, baterie a síťové komunikace. Jedná se opravdu o užitečnou funkci spíše novějších (v době psaní této práce) chytrých telefonů, díky které se zvyšuje výdrž zařízení. V případě aplikací zpracovávající krátkodobé a nenáročné operace se o tak závažný problém nejedná, spíše naopak. Problém spočívá v tom, že tato funkce v základním *Androidu* neexistuje a přidávají ji až samotní výrobci jednotlivých telefonů. To ale zároveň znamená, že tuto situaci nelze předem ovlivnit. Řešení by spočívalo v nastudování platforem všech výrobců, kteří si ale příliš nelámou hlavu s dokumentací. Situace je ještě o to horší, kdy zařízení zakáže získávání bodů pomocí *GPS*, ale z pohledu aplikačního rozhraní *Androidu* je *GPS* stále aktivní. Pokud chceme této situaci předejít, jsou v zásadě celkem dvě metody:

- **Výjimka aplikace** – povolení výjimky pro danou aplikaci v nastavení telefonu,
- **Uzamčení aplikace** – uzamčení aktivity v seznamu právě spuštěných aplikací po dobu měření jízdy.

Ani v jednom případě se ale nejedná o způsob, který by byl z pohledu nezkušeného uživatele triviální. Ve výsledku lze uživatele pouze upozornit, že tato situace může nastat, nebo jej informovat pomocí notifikace, že pravděpodobně k této situaci došlo. Pravděpodobně proto, že se může jednat i o pouhou ztrátu signálu. Při testování bylo zjištěno, že v momentě, kdy zařízení po několika desítkách sekund nedostane žádnou souřadnici, lze *API* restartovat a poté nedojde ani k tzv. *first fix* satelitů. Pro implementaci kontroly *listenerů* byl využit tzv. *TimerTask*, který umožňuje na straně služby cyklicky kontrolovat stav satelitů a čas poslední souřadnice, poté je případně vytvořen nový segment.

Méně komplikovaný problém spočívá při kontrole zapnutí *GPS* v zařízení, který lze předem zkontrolovat a informovat uživatele. Opět v návaznosti na konkrétním výrobci, je někdy nutné právo k lokalizaci pomocí satelitů přidělit jednotlivým aplikacím. To má za následek, že i přes zapnuté *GPS* může aplikace požadovat aktivaci.

10.2.4 Synchronizace jízd

Synchronizace jízd mezi mobilním zařízením a serverovou aplikací je prováděno automaticky pomocí registrovaného *Broadcast receiveru* (třída `ServerTrackReceiver`) v časových periodách nebo manuálně tažením od horního kraje fragmentu `RouteListFragment`.

Za samotnou synchronizaci odpovídá třída `Synchronization` v balíčku `SocketIO`, která implementuje abstraktní třídu `ASocketIO`. Voláním metody `connect` je provedeno připojení k serverové aplikaci, které je bezprostředně následováno autentizací popsané v kapitole 10.1.5. V případě úspěchu, metoda `onConnect` spustí nejprve odeslání nově vytvořených jízd a následně stažení nových nebo aktualizovaných jízd.

Veškeré jízdy jsou na server odesílány postupně s čekáním na potvrzení úspěšného uložení. Jízda je emitována zprávou `phone_synchronization_upload`, kde data jsou ve formátu *JSON* podle uvedeného schématu v návrhu 8.5. Každá zpráva má nastavený *timeout* pomocí metody `Timer.schedule`, který po vypršení označí celou synchronizaci za neúspěšnou a spojení se serverem je ukončeno. V opačném případě, po úspěšném odeslání všech nových jízd následuje přijmutí.

Serverová aplikace před odesláním všech nových/upravených jízd nejprve emituje zprávu `phone_synchronization_download` obsahující počet jízd, které má mobilní zařízení očekávat. Následně jsou přijímány zprávy `phone_synchronization_download_track` obsahující objekt jízdy ve formátu *JSON* podle definovaného schématu v kapitole 8.5.

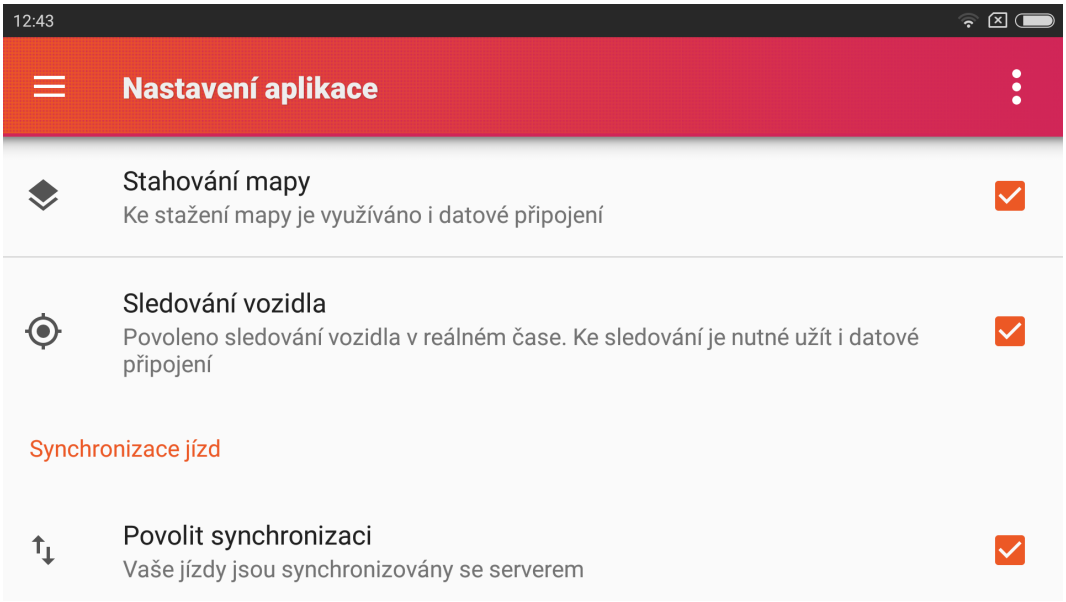
Pro sestavení příchozího objektu ve formátu *JSON* je využita knihovna *Gson*[5]. Metoda `Gson.fromJson(String, Track.class)` zpracovává textový řetězec ve formátu *JSON* a konvertuje jej do instance třídy definované druhým parametrem. V případě, že serverová aplikace pošle nesprávná data, metoda zareaguje výjimkou `JsonSyntaxException`. V tomto případě je spojení se serverem ukončeno a synchronizace označena jako neúspěšná.

10.2.5 Nastavení aplikace

Fragment `SettingsFragment`, na rozdíl od ostatních, je potomkem abstraktní třídy `PreferenceFragmentCompat`, která se stará o vzhled uživatelského prostředí pomocí definice souboru `fragment_settings.xml`. Třída zároveň dědí od třídy `Fragment`, tudíž je možné jej klasicky zobrazit v aktivitě.

Soubor *XML* definuje elementy `CheckBoxPreference` obsahující atribut `key`, který uchovává klíč pro uložení hodnoty typu `boolean` do již zmíněného *Shared Preferences*.

Podle uživatelského nastavení se řídí některé části zdrojového kódu. Uživatel konkrétně může ovlivnit stahování mapových podkladů pomocí datového připojení, sledování vozidla v reálném čase, povolení synchronizace a synchronizování pomocí datového připojení.



Obrázek 10.7: Fragment nastavení aplikace.

[5] https://github.com/google/gson

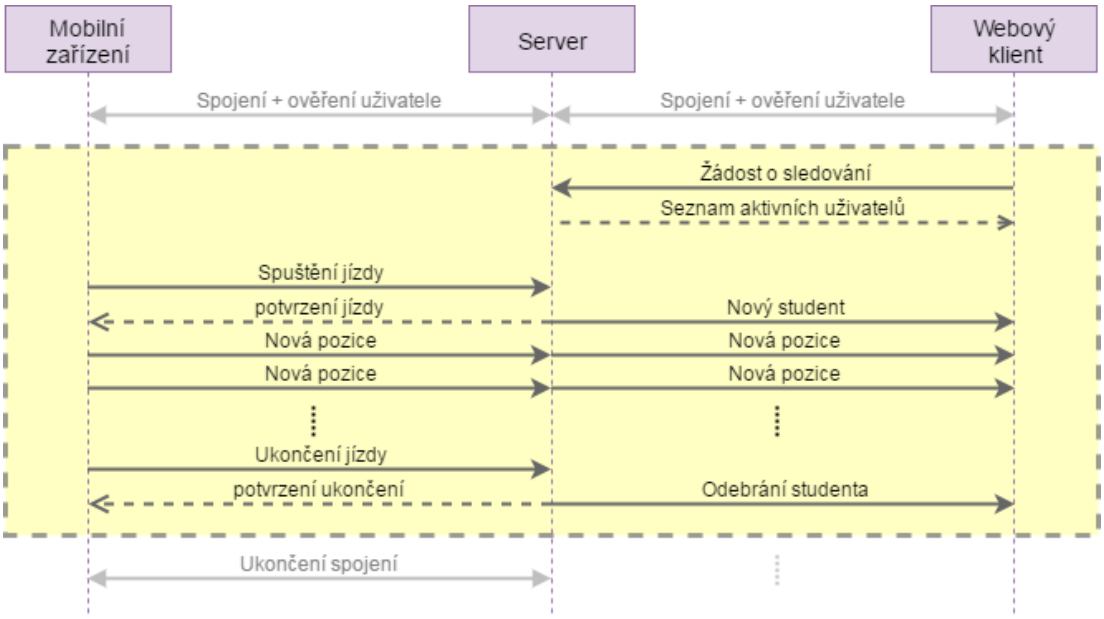
10.2.6 Logování stavu aplikace

Pořizování *log* výpisů probíhá se stejnými pravidly jako u serverové aplikace zmíněné v kapitole 10.1.8. Podmínkou je mít připojené mobilní zařízení k *PC*, na kterém probíhá vývoj aplikace pomocí *Android Studia*. V momentě testování zařízení v reálném provozu je tento způsob nedostačující, protože výpis není nikde vidět a ani se nikam neukládá. Z tohoto důvodu jsem využil knihovnu *ACRA*[6], která umožňuje vzniklá chybová hlášení odeslat na e-mailovou adresu. V případě potřeby umožňuje i ukládání všech hlášení do externí databáze *MongoDB*, které jsem nakonec nevyužil, protože testování v reálném provozu probíhalo až s funkčně odladěnou aplikací.

10.3 Sledování vozidel v reálném čase

Návrh struktury systému pro on-line záznam jízd a sledování v reálném časem byl uveden v kapitole 8.1. Z kapitol 10.2 a 10.1 je čtenáři známa problematika a princip implementace připojení klientů, jejich autentizace, rozdělení do místností a samotná komunikace v reálném čase. V této části textu bude popsán princip záznamu a sledování jízdy v reálném čase z pohledu tří stran, tedy mobilní aplikace snímající jízdu, serverová aplikace zpracovávající příchozí pozice a webový klient, který sleduje jízdy studentů v rámci konkrétní autoškoly. Tyto aktéry je možné vidět na obrázku 10.8, který demonstruje komunikaci v reálném čase.

Z uvedených kapitol víme, že každý uživatel mobilního zařízení může být přihlášen k serverové aplikaci pouze jedinkrát. Naopak uživatel webového klienta může být autentizován vícekrát. Schéma komunikace na obrázku 10.8 je pro zjednodušení situace vyobrazen pouze pro jedno mobilní zařízení a jeden webový klient. Princip komunikace bude dále vysvětlen stejným způsobem. Je ale zřejmé, že v reálném použití bude v systému figurovat více klientů.



Obrázek 10.8: Schéma komunikace při sledování jízdy v reálném čase.

[6] https://github.com/ACRA/acra

V momentě, kdy je webový klient úspěšně autentizován pomocí zmíněné výměny zpráv k serverové aplikaci, emituje zprávu `web_real_time_users` žádající o získání seznamu aktivních studentů. Serverová aplikace vrátí v *callback* funkci pole objektů typu `RealTimeStudent`, který je definován na obrázku 10.9.

```
TRIP = {
  sid: Number,
  uid: Number,
  gid: Number,
  aid: Number,
  timeStart: Number,
  points: [GeoPoint]
}

REALTIMESTUDENT = {
  student: Student,
  mongoRealTimeID: String,
  socketID: String,
  trip: TRIP
}

STUDENT = {
  id: Number,
  ended: Number,
  company_id: Number,
  group_id: Number,
  user_id: Number,
  company_name: String,
  group_name: String,
  group_shcut: String,
  user_name: String
}
```

Obrázek 10.9: Formát objektů ve formátu *JSON* pro *real-time* komunikaci.

Mobilní zařízení začíná snímat jízdu odesláním zprávy `phone_realtime_start_track` s objektem obsahující čas začátku a identifikátor zvoleného kurzu (v případě ztráty síťového připojení je součástí objektu i identifikátor databáze *MongoDB* kvůli znovunavázání). Serverová aplikace vytvoří nový záznam v databázi funkcí `realtime_createNewTrip` (součástí *API*), která v *callback* funkci vrací identifikátor nově uložených dat. Serverová aplikace tento identifikátor vrátí přes aktivní *socket* zpět a celé skupině připojených autoškol je zaslána zpráva `web_real_time_start` s objektem typu `RealTimeStudent`.

Obrázek 10.10: Sledování chytrého telefonu pomocí webového prohlížeče.

V tento moment mobilní zařízení začne při získání nové pozice odesílat zprávu `phone_realtime_new_location` včetně objektu typu `GeoPoint`. Serverová aplikace následně upraví souřadnici podle reálné vozovky, viz následující kapitola 10.4, a přihlášeným uživatelům dané autoškoly pošle zprávu `web_real_time_location` s upraveným objektem typu `GeoPoint`.

Ukončení jízdy probíhá obdobným způsobem jako zahájení jízdy, kde mobilní zařízení odešle zprávu `phone_realtime_stop_track`, server odpoví a následně emituje zprávu `web_real_time_stop` směrem ke klientovi. Při dokončení je *real-time* jízda smazána z databáze a je nutná synchronizace pro uchování nové jízdy studenta. Důvodem je, že při

on-line snímání jsou z důvodu menšího objemu dat odesílány pouze vyfiltrované souřadnice. Navíc jízda obsahuje při svém dokončení další statistické údaje. Tyto informace je nutné synchronizovat se serverovou aplikací pro další analýzy.

10.4 Editování tras

10.4.1 Automatická editace

Automatické editování tras probíhá bezprostředně po odeslání jízdy na server nebo po manuální editaci jízdy uživatelem. Úpravu zprostředkovává třída `Matching`, která je součástí serverové aplikace v balíčku `ServerApps/MapData`. Třída obsahuje tři veřejné metody:

- **snapTrackToRoad(_id, callback)** – vstupním parametrem je identifikátor typu `string` jízdy a anonymní funkce vracející objekt typu `Track`, který obsahuje parametr `segmentsEdit` obsahující pole upravených segmentů,
- **snapSegmentsToRoad(_id, segment, callback)** – vstupním parametrem je řetězcový identifikátor jízdy, objekt typu `ISegment` (obecný segment) a anonymní funkce vracející pole objektů typu `SegmentEdit`, které reprezentuje upravené segmenty,
- **saveSegmentsEdit(_id, segments, callback)** – vstupním parametrem je řetězcový identifikátor jízdy, pole objektů typu `SegmentEdit` a anonymní funkce vracející pravdivostní parametr `saved`, který informuje o úspěchu uložení upravených segmentů.

Pro zaslání *HTTP* požadavku běžící aplikaci *OSRM* je využit balíček *Request*[7], který je stažitelný z *npm*. Konkrétní příklad je znázorněný algoritmem 10.3.

```
// Odeslání požadavku
Request({url: strUrl, json: true}, function(error,response,data){
    // Zpracování výsledku OSRM
});
```

Algoritmus 10.3: Zaslání požadavku aplikaci *OSRM*.

URL musí splňovat definovaný formát 10.3, kde `coordinates` definuje souřadnice oddělené středníkem ve formátu *délka, šířka*. *URL* obsahuje další nepovinné parametry, jako například `overview` určující kvalitu přimknutí, `timestamps` určující časové hodnoty souřadnic oddělené čárkou a `radiuses` určující přesnost souřadnic získané pomocí mobilního zařízení.

$$http://IP:PORT/match/v1/driving/\{coordinates\}?geometries=geojson \quad (10.3)$$

Výsledkem operace *OSRM* je posloupnost upravených souřadnic ve formátu *JSON*, kde souřadnice je definována jako pole `[délka,šířka]`. Přesné rozhraní[8] je definováno v souboru `IOSRM`.

Výsledkem operace „přimknutí“ *OSRM* nemusí být vždy ten samý segment. Například v případě zastaralých mapových podkladů, algoritmus nemusí nalézt správnou cestu.

[7] https://www.npmjs.com/package/request
[8] https://github.com/Project-OSRM/osrm-backend/blob/master/docs/http.md

V takovém případě je výsledkem rozdělení segmentu do více segmentů, které je nutné uložit do databáze pomocí funkce `updateSegmentsEdit` (součástí *API MongoDB*).

Obrázek 10.11: Srovnání originální a upravené trasy.

Pro výpočet nejbližší souřadnice při snímání jízdy v reálném čase je opět použito *OSRM*. Tuto operaci zprostředkovává třída `Nearest`, která má jedinou veřejnou metodu `nearestGeoPoint` zpracovávající parametr typu *IGeoPoint* a anonymní funkci vracející upravený bod. Pro zaslání požadavku je opět použit algoritmus 10.3. Jediný rozdíl spočívá ve formátu 10.4 url.

$$http://IP:PORT/nearest/v1/driving/\{longitude, latitude\} \tag{10.4}$$

Serverová aplikace využívá komunikaci s *OSRM*, která je plnohodnotně otestována v rámci lokálního použití, ale poskytovatel produkčního serveru v čele s vedoucím projektu *DoAutoškoly.cz* doposud nezajistil spuštění aplikace. Tento nedostatek ve výsledku neovlivňuje používání serverové aplikace, která prozatím běží na testovacím serveru *DoAutoškoly.cz*. Trasy jsou nadále synchronizovány a uživatelem upravovány, ale nedochází k automatickému přimknutí podle reálné vozovky.

10.4.2 Manuální editace

Manuální editování jízdy probíhá formou implementování modulu pro stávající portál *DoAutoškoly.cz*. Pro vykreslování a editování tras na mapovém podkladu byl využit balíček *Leaflet*, který nabízí jednoduché aplikační rozhraní. Vytvoření mapového podkladu nastiňuje algoritmus 10.4.

```
// Vytvoření mapy
var map = L.map("#identifikátor");

// Zavedení mapových podkladů
var maps : L.Control.LayersObject = {
    'Základní': L.tileLayer(...),
    'Letecká':  L.tileLayer(...)
}

// Přidání vrstev do mapy
L.control.layers(maps, {}, {}).addTo(map);
```

Algoritmus 10.4: Vytvoření mapy s využitím knihovny *Leaflet*.

Pro vytvoření trasy je nutné vytvořit objekt typu `L.Polyline`, který je následně přidán do mapy pomocí funkce `addLayer`. Tuto funkci zapouzdřuje objekt typu `L.FeatureGroup`.

Všechny segmenty jsou při vykreslení sloučeny do jednoho tak, aby uživatel byl nucený svoji jízdu upravit plnohodnotně. Zároveň uživatel má zakázáno vykreslovat nové objekty. Knihovna *Leaflet* umožňuje registrovat *listener* (algoritmus 10.5), který zprostředkuje emitování zprávy `web_tracks_editing_track_send` obsahující objekt typu `SegmentEdit`.

```
// Listener - dokončení editování vrstvy
map.on(L.Draw.Event.EDITED, function(e : any) {
    e.layers.eachLayer(function(layer) {
        // vždy jedna vrstva
        // zpracování upravené polyline
    });
});
```

Algoritmus 10.5: Vytvoření listeneru s využitím knihovny *Leaflet*.

10.5 Zabezpečení

Při implementaci aplikace typu *klient-server* je kromě samotné funkčnosti velmi důležitá také bezpečnost síťové komunikace. K bezpečnosti patří napadení celé služby, krádež dat, nevhodná změna dat, apod. Obvyklým problémem je, že uživatelé používají stejné heslo pro několik služeb. Odcizení takového hesla může mít fatální následky. V tomto případě by útočník mohl získat přístup k velmi citlivým údajům, konkrétně k poloze jednotlivým uživatelům v reálném čase. Ukládání citlivých dat v chytrém zařízení platformy *Android* bylo popsáno v kapitole 10.2.2. V následující části bude popsána bezpečnost čistě z pohledu síťové komunikace s využitím rámce *Socket.IO*.

Veškerá síťová komunikace je šifrována s využitím symetrické blokové šifry *AES*, která k šifrování i dešifrování používá stejný klíč. Šifrování obstarává třída `Crypto`, která využívá balíček *crypto-js*, stažitelný z *npm*. Sestavený objekt, který bude emitován z klienta na server nebo naopak, je nejprve převeden do textového řetězce. Ten je pomocí funkce `encryptAES` zašifrován a následně odeslán. Na druhé straně je řetězec rozšifrován metodou `decryptAES` a poté převeden z textové podoby do konkrétního objektu.

Samotné šifrování komunikace nemusí stačit. Z důvodu útoku *Man-in-the-middle* je vhodné použít zabezpečený přenos pomocí *HTTPS*, který umožňuje i framework *Socket.IO*. Jediný rozdíl ve vytvoření serverové aplikace spočívá v importování modulu *https* namísto *http* plus použití modulu *fs*, který umožňuje načíst potřebné certifikáty. Z pohledu klienta stačí změnit protokol adresy na *https* a v nastavení použít parametr `secure` s hodnotou `true`.

Tento druh zabezpečení nebyl prozatím zaveden, protože serverová aplikace dosud běží na testovacím serveru poskytovatele, kde certifikát není pořízen. Naopak při ostrém spuštění serverové aplikace je nutné certifikát importovat.

Kapitola 11

Testování

Základní testování celého systému typu *klient-server* probíhalo prakticky po celou dobu vývoje. V první fázi realizace byla vyvíjena zejména mobilní aplikace, která je z hlediska testování nejkomplikovanější. A to zejména z důvodu dlouhotrvající služby, která využívá *GPS* a síťové připojení. Každá verze platformy *Android*, a i v závislosti na výrobci, reaguje rozdílným způsobem a je nutné zařízení správně nastavit. V první fázi byl vytvořen jednoduchý prototyp sloužící pouze k snímání jízdy bez další funkčnosti, jako je například přihlášení uživatelů, které si žádá síťovou komunikaci. Důvodem bylo odstranit počáteční elementární chyby v rámci snímání a zjistit chování jednotlivých dostupných verzí platformy *Android*. Tento prototyp byl dodán několika uživatelům s rozdílnými verzemi a výrobci zařízení. V této fázi byly zjištěny jak poznatky negativní (popsané v kapitole 10.2.3), tak i pozitivní. Tento prototyp dále sloužil jako základ pro rozšiřování a zdokonalování mobilní aplikace pomocí získaných poznatků od uživatelů až do finální podoby, která bude ukázána v příloze.

Druhou fází byla implementace serverové aplikace, kde bylo nejprve nutné realizovat synchronizování naměřených jízd uživatelů s databází *MongoDB* pro účely dalšího testování, i v návaznosti s diplomovou prací Bc. Martina Šouláka. Vzhledem k chybějícímu nástroji pro správu databáze *MongoDB* byl vytvořen jednoduchý webový klient, který umožňoval výpis stavu databáze a vykreslení jízd podle jejich identifikátoru pomocí *Open Street Map*.

Poslední fáze znamenala implementaci a testování komunikace v reálném čase společně se snímáním jízdy, kde již byla k dispozici funkčně plnohodnotná mobilní aplikace. Testování bylo zprvu prováděno čistě pomocí jediného mobilního zařízení v rámci lokální sítě a záměrného náhodného rozptýlení souřadnic. V další fázi byly využívány jakékoliv dopravní prostředky nevyjímaje kolejových vozidel v reálném provozu, kde bylo možné zároveň snímat jízdu a sledovat chování na druhém zařízení přes serverovou aplikaci. V další a zároveň poslední fázi byla aplikace rozšířena mezi skupinu testovacích uživatelů, kteří byli ochotni snímat jízdy a nechat se sledovat. Díky těmto získaným údajům bylo otestováno skutečně reálné sledování provozu.

Výsledkem testování je 140 naměřených jízd, které jsou uchovány v databázi *MongoDB*. Do tohoto počtu nejsou započítány veškeré jízdy, které z důvodu pozměnění schématu databáze byly nuceně odstraněny. Jízdy v databázi jsou různého charakteru. Objevují se zde trasy, které mají v průměru 3000 až 6000 geografických bodů, ale taky i jízdy mající maximálně 1000 geografických bodů. S každou zaznamenanou jízdou se zároveň zvyšovalo otestování systému včetně aplikace chytrého telefonu a implementovaných modulů pro webový prohlížeč.

Kapitola 12

Závěr

Úvodní kapitola nabídla seznámení s portálem *DoAutoškoly.cz* zabývající se problematikou autoškol v ČR. Projekt Ing. Jana Hrivnáka je stále ve svých počátcích, ale osobně si myslím, že je velmi zajímavý a snaží se prosadit v tématu, které zajímá každého z nás. Právě díky tomu, že se jedná o začínající projekt, nabízí bohaté možnosti k přispění v rozvoji.

Osobně jsem velmi rád, že jsem mohl přispět a seznámit se s tímto projektem. Přestože moje bakalářská práce se taktéž zabývala platformou *Android*, tak mi tato práce přinesla množství nových zkušeností v problematice. Osobně považuji za důležité se neustále seznamovat s novými technologiemi, mezi které určitě spadá kombinace *Node.js*, *React.js* a *Socket.IO* včetně databází *NoSQL* umožňující realizaci zajímavých projektů.

První kapitoly jsou spíše teoretického významu a slouží pro pochopení problematiky. Kapitola 8 na základě teoretické části a formálního zadání práce nastiňuje návrh aplikace typu *klient-server* včetně elementárních částí. Následující kapitoly 10.2 a 10.1 popisují princip implementace aplikace pro platformu *Android*, resp. serverové aplikace. Pro témata sledování v reálném čase 10.3, editování tras 10.4 a zabezpečení 10.5 byly vyčleněny samostatné kapitoly. Důvodem je prolínání principů při vývoji těchto stěžejních částí systému. Naopak v případě modulu pro webový katalog nebyla uvedena vlastní kapitola, protože realizace vyplývá z volby technologií v kapitole 9.2, principu autentizace k serverové aplikaci 10.1.5 a dále již zmíněné kapitoly popisující *real-time* komunikace a editování tras. Přesné návody pro spuštění či instalaci aplikací budou uvedeny na přiloženém disku.

Z pohledu implementace byly splněny všechny formální a motivační cíle z kapitoly 3. Spuštění celého systému do reálného provozu proběhne v závislosti na rozhodnutí autora projektu *DoAutoškoly.cz*. Veškeré implementované části jsou realizované tak, aby je bylo možné podle návodu nasadit do produkčního prostředí. Podobně i v případě mobilní aplikace, která by měla být přístupná z oficiálního *storu*. Současným cílem vedoucího projektu Ing. Jana Hrivnáka je po dohodě s konkrétními autoškolami dodat mobilní aplikaci lektorům, kteří budou zprvu provádět snímání jízd. Po nasbírání určitého množství dat a získání zpětné vazby bude aplikace určena ke stažení studentům.

Jednotlivé součásti celého systému byly navrhnuty a implementovány s cílem budoucí rozšiřitelnosti, která se jistě v projektu *DoAutoškoly.cz* nabízí. V případě úspěšného nasazení systému lze vytvořit mobilní aplikaci pro platformu *iOS*. Další velmi zajímavou možností je vytvoření dalších analýz naměřených tras, jako je například analýza projetých křižovatek, směr odbočení apod. Tuto možnost nabízí přímo zmíněné *OSRM*, které při výpočtu přimknutí definuje, mimo jiné, i souřadnice jednotlivých křižovatek.

Literatura

[1] ČADA, V.: *Souřadnicové systémy*. [Online; navštíveno 6.12.2016].
URL http://gis.zcu.cz/studium/gen1/html/ch02s03.html

[2] ČADA, V.: *Závazné souřadnicové systémy v ČR*. [Online; navštíveno 5.12.2016].
URL http://www.zememeric.cz/csgk/apig99/referaty/ref_4.htm

[3] ALLEN, G.: *Android 4 - Průvodce programováním mobilních aplikací*. Computer Press, 2013, ISBN 978-80-251-3782-6.

[4] CAPAN, T.: *Why The Hell Would I Use Node.js*. [Online; navštíveno 9.4.2017].
URL https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js

[5] ŠEBESTA, J.: *Globální navigační systémy*. FEKT VUT v Brně, 2012, ISBN 978-80-214-4500-0.

[6] *Reconciliation*. [Online; navštíveno 11.4.2017].
URL https://facebook.github.io/react/docs/reconciliation.html

[7] *Activity*. [Online; navštíveno 4.12.2016].
URL https://developer.android.com/reference/android/app/Activity.html

[8] *Dashboards*. [Online; navštíveno 4.12.2016].
URL https://developer.android.com/about/dashboards/index.html

[9] *LocationListener*. [Online; navštíveno 23.12.2016].
URL https://developer.android.com/reference/android/location/LocationListener.html

[10] *LocationManager*. [Online; navštíveno 23.12.2016].
URL https://developer.android.com/reference/android/location/LocationManager.html

[11] *Location Strategies*. [Online; navštíveno 7.12.2016].
URL https://developer.android.com/guide/topics/location/strategies.html

[12] *Fundamentals of Mapping*. [Online; navštíveno 31.3.2017].
URL http://www.icsm.gov.au/mapping/index.html

[13] *Global Positioning System - Standart Positioning Service Signal Specification*. [Online; navštíveno 4.12.2016].
URL http://www.gps.gov/technical/ps/1995-SPS-signal-specification.pdf

[14] HAVLENA, M.: *Node.js – How it differs from traditional web servers.* [Online; navštíveno 9.4.2017].
URL http://www.havlena.net/en/programming/node-js-how-it-differs-from-traditional-web-servers/

[15] HOFMANN-WELLENHOF, B.; LICHTENEGGER, H.; COLLINS, J.: *Global Positioning System: Theory and Practice.* Wien: Springer-Verlag, páté vydání, 2001, ISBN 978-3-7091-6199-9.

[16] HRIVNÁK, J.: *O projektu DoAutoškoly.cz.* [Online; navštíveno 1.12.2016].
URL https://www.doautoskoly.cz/o-projektu/

[17] HRIVNÁK, J.: *Zvyšování kvality autoškol pomocí sdílení uživatelských zkušeností.* Diplomová práce, FIT VUT v Brně, 2016.

[18] HRUBÝ, M.: *Geografické Informacní Systémy.* [Online; navštíveno 5.12.2016].
URL http://perchta.fit.vutbr.cz/vyuka-gis/uploads/1/GIS-final2.pdf

[19] JATI, A.: *Content Providers in Android.* [Online; navštíveno 4.12.2016].
URL
http://www.oodlestechnologies.com/blogs/Content-Providers-in-Android

[20] KYPTA, T.: *Vyvíjíme pro Android.* [Online; navštíveno 4.12.2016].
URL http://www.abclinuxu.cz/clanky/vyvijime-pro-android-uvod

[21] LACKO, L.: *Vývoj aplikací pro Android.* Computer Press, 2015, ISBN 978-80-251-4347-6.

[22] LUBBERS, P.; GRECO, F.: *HTML5 WebSocket: A Quantum Leap in Scalability for the Web.* [Online; navštíveno 2.4.2017].
URL https://www.websocket.org/quantum.html

[23] LUXEN, D.; VETTER, C.: *Real-time routing with OpenStreetMap data.* In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '11, New York, NY, USA: ACM, 2011, ISBN 978-1-4503-1031-4, s. 513–516, doi:10.1145/2093973.2094062.
URL http://doi.acm.org/10.1145/2093973.2094062

[24] MALÝ, M.: *Kometa přináší web v reálném čase.* [Online; navštíveno 3.4.2017].
URL https://www.zdrojak.cz/clanky/kometa-prinasi-web-v-realnem-case/

[25] MALÝ, M.: *Web Sockets.* [Online; navštíveno 3.4.2017].
URL https://www.zdrojak.cz/clanky/web-sockets/

[26] MELNIKOV, A.: *The WebSocket Protocol.* Google, Inc., 2011, ISSN 2070-1721.
URL https://tools.ietf.org/html/rfc6455

[27] ORLICH, M.: *Základní lokalizační metody v GSM.* [Online; navštíveno 7.12.2016].
URL http://access.feld.cvut.cz/view.php?cisloclanku=2006022801

[28] RAHMAN, M. Z.: *Beyond Trilateration: GPS Positioning Geometry and Analytical Accuracy.* InTech, 10 vydání, 2012, ISBN 978-953-307-843-4.

[29] RAPANT, P.: *Geoinformatika a geoinformacní technologie*. Ostrava: VŠB Technická univerzita Ostrava, 2006, ISBN 80-248-1264-9.

[30] RUSSELL, A.: *Comet: Low Latency Data for the Browser*. [Online; navštíveno 1.4.2017].
URL
http://infrequently.org/2006/03/comet-low-latency-data-for-the-browser/

[31] *About SQLite*. [Online; navštíveno 5.12.2016].
URL https://sqlite.org/about.html

[32] VOŽENÍLEK, V.: *Aplikovaná kartografie I.: tématické mapy*. Olomouc: Univerzita Palackého v Olomouci, 2001, ISBN 80-244-0270-x.

[33] *Usage of client-side programming languages for websites*. [Online; navštíveno 10.4.2017].
URL https://w3techs.com/technologies/overview/client_side_language/all

[34] *About HTML5 WebSocket*. [Online; navštíveno 2.4.2017].
URL https://www.websocket.org/aboutwebsocket.html

[35] WICKRAMASINGHE, P.: *SignalR – Real-time application development*. [Online; navštíveno 2.4.2017].
URL http://developereventlog.blogspot.cz/2013/06/aspnet-signalr-real-time-application_6.html

[36] WILLAREDT, J.: *WiFi and Cell-ID based positioning - Protocols, Standards and Solutions*. [Online; navštíveno 7.12.2016].
URL https://www.snet.tu-berlin.de/fileadmin/fg220/courses/WS1011/snet-project/wifi-cellid-positioning_willaredt.pdf

Přílohy

Příloha A

Obsah přiloženého CD

Přiložený disk k této diplomové práci obsahuje tyto soubory:

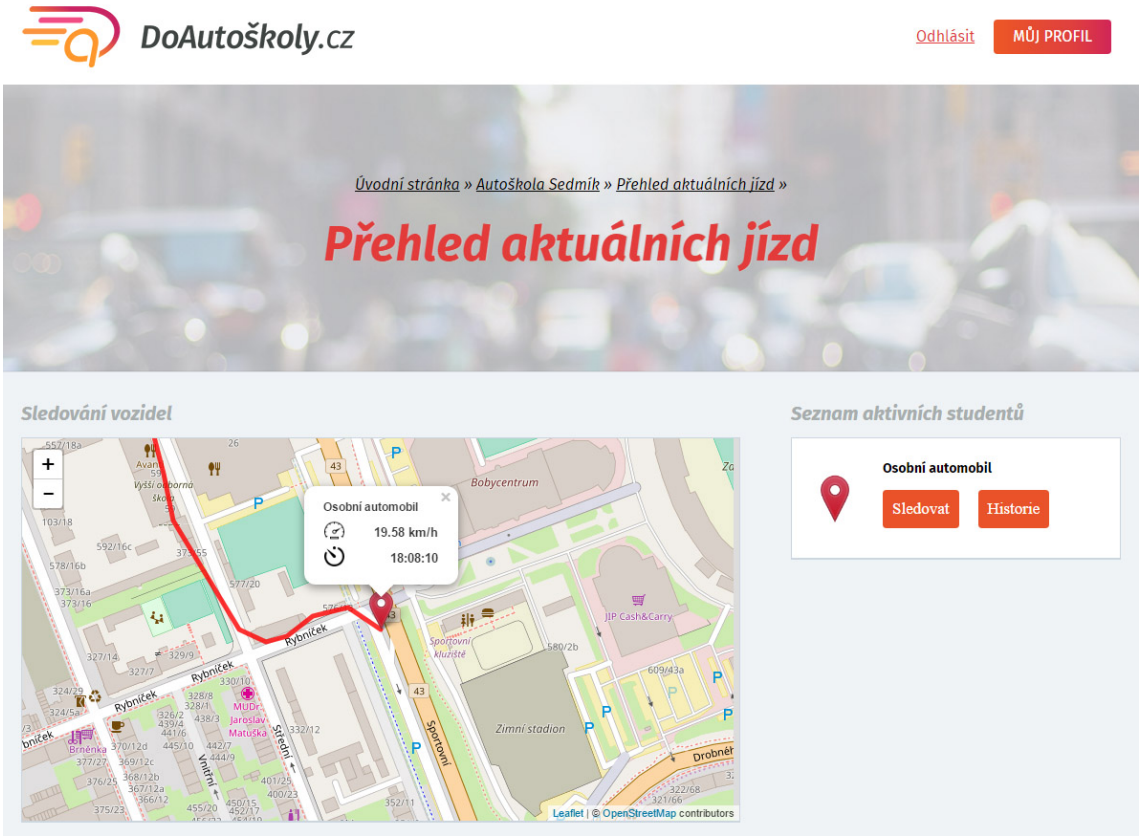
- **Snimky/** – uchovává snímky vytvořené mobilní aplikace a obou webových modulů,
- **TechnickaZprava/** – uchovává zdrojové soubory k sestavení této technické zprávy,
- **ZdrojoveSoubory/**
  - **ClientskaAplikace/** – obsahuje příslušné zdrojové kódy webového modulu pro portál *DoAutoškoly.cz* a soubor `README` obsahující návod k sestavení a spuštění aplikace,
  - **MobilniAplikace/** – obsahuje příslušné zdrojové kódy mobilní aplikace platformy *Android* a soubor `README` obsahující návod k sestavení a spuštění aplikace,
  - **ServerovaAplikace/** – obsahuje příslušné zdrojové kódy serverové aplikace a soubor `README` obsahující návod k sestavení a spuštění aplikace,
- **README.txt** – textový soubor s popisem obsahu přiloženého CD,
- **TechnickaZprava.pdf** – soubor ve formátu *.pdf* obsahující tuto technickou zprávu.

Příloha B

Ukázky aplikace

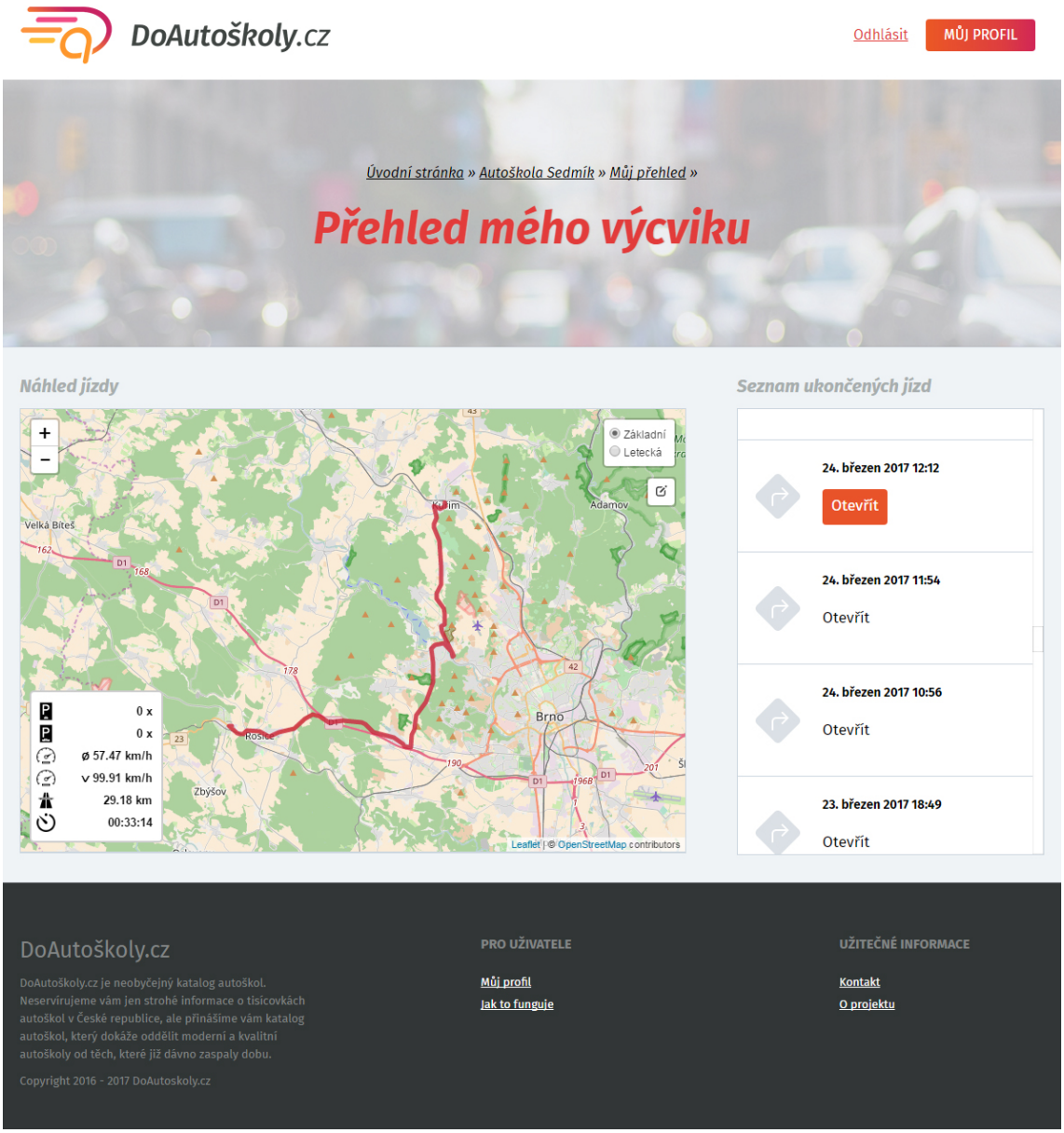
V této příloze budou dále uvedeny ukázky z jednotlivých klientských aplikací, které spolu komunikují. Konkrétně se jedná o webový modul pro majitele autoškoly, webový modul pro studenta autoškoly, a nakonec mobilní aplikace sloužící k měření a zaznamenávání jízd autoškoly. Moduly určené pro zobrazení pomocí webového prohlížeče jsou zasazeny do cíleného webového portálu *DoAutoškoly.cz*.

B.1 Webový modul – majitel

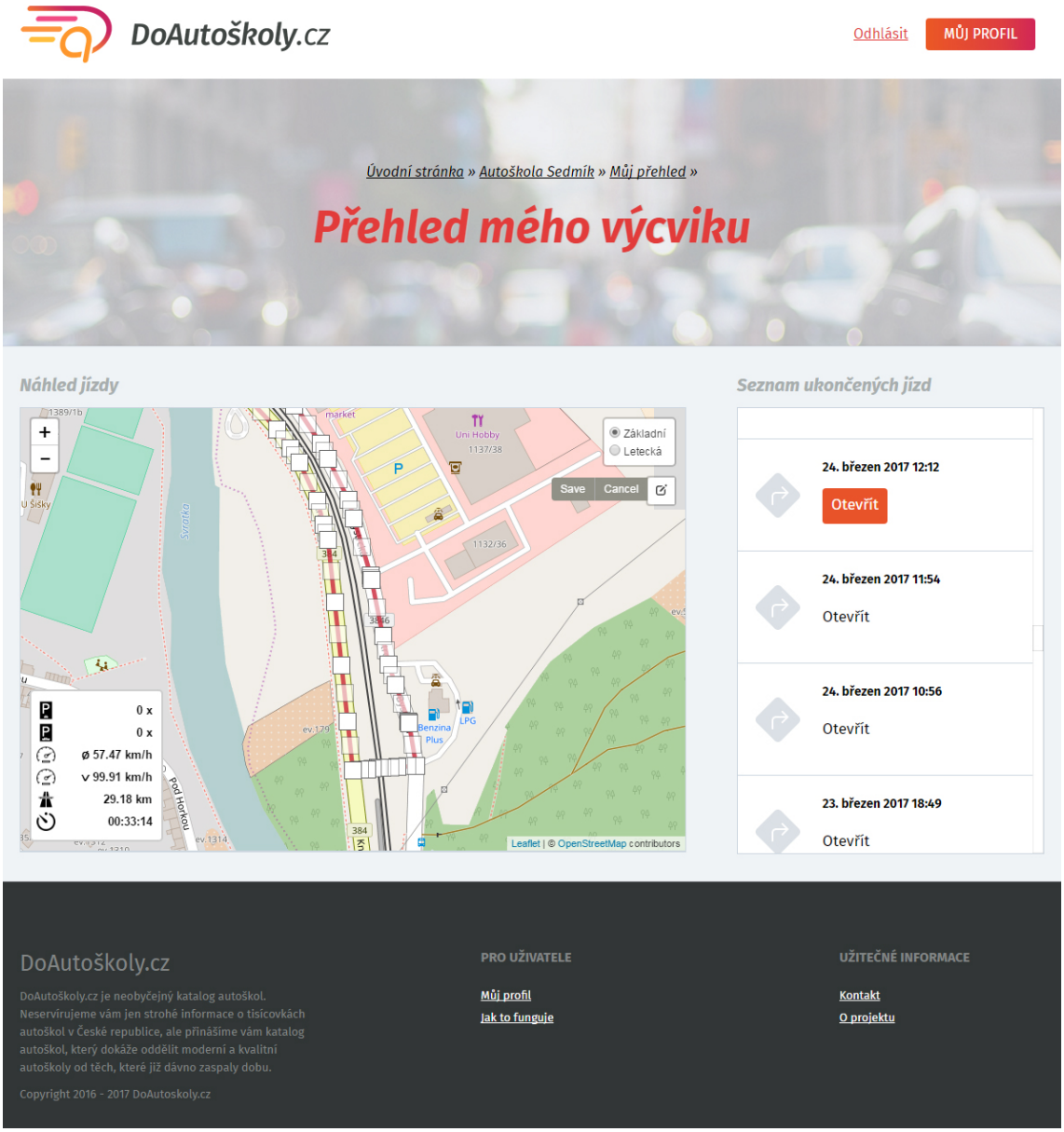


Obrázek B.1: Ukázka webového modulu sloužícího pro sledování studentů dané autoškoly.

B.2 Webový modul – student



Obrázek B.2: Ukázka webového modulu sloužícího pro prohlížení a editování naměřených tras studenta. Konkrétně na tomto obrázku je znázorněna celá naměřená trasa s vyobrazenými statistikami. V pravém boxu je vyobrazen posuvný seznam s ukončenými jízdami.

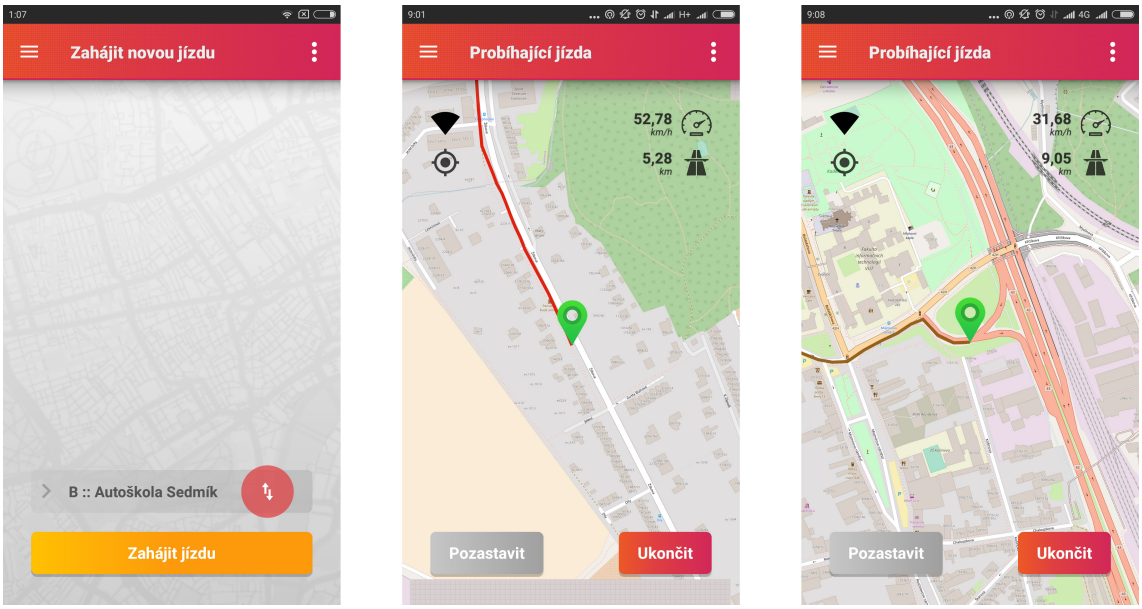


Obrázek B.3: Ukázka webového modulu sloužícího pro prohlížení a editování naměřených tras studenta. Konkrétně na tomto obrázku je znázorněn výřez trasy, který je možné upravit. Tažením za bílý bod trasy uživatel provede posunutí, podobně jako u průhledného bodu, u kterého se navíc při posunutí vytvoří z obou stran po jednom bodu. Kliknutím na bílý bod se provede odstranění souřadnice.

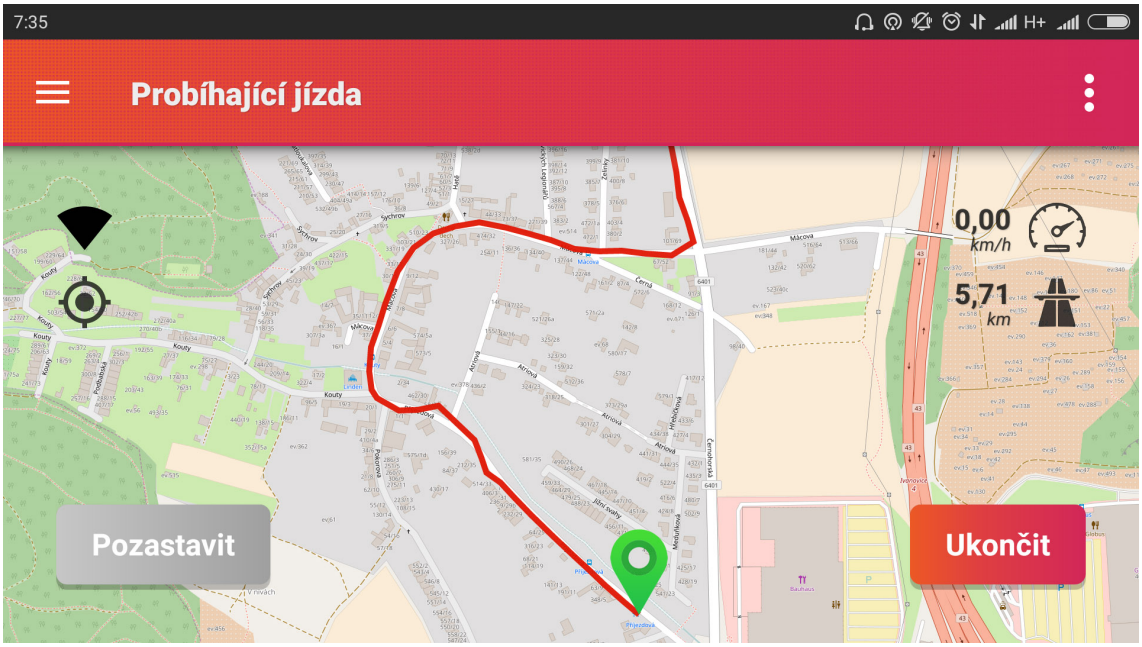
B.3 Automatická úprava jízd

Obrázek B.4: Obrázek demonstruje výstup *OSRM* – konkrétní případy automatického přimknutí naměřené trasy pomocí *GPS* podle reálné komunikace.

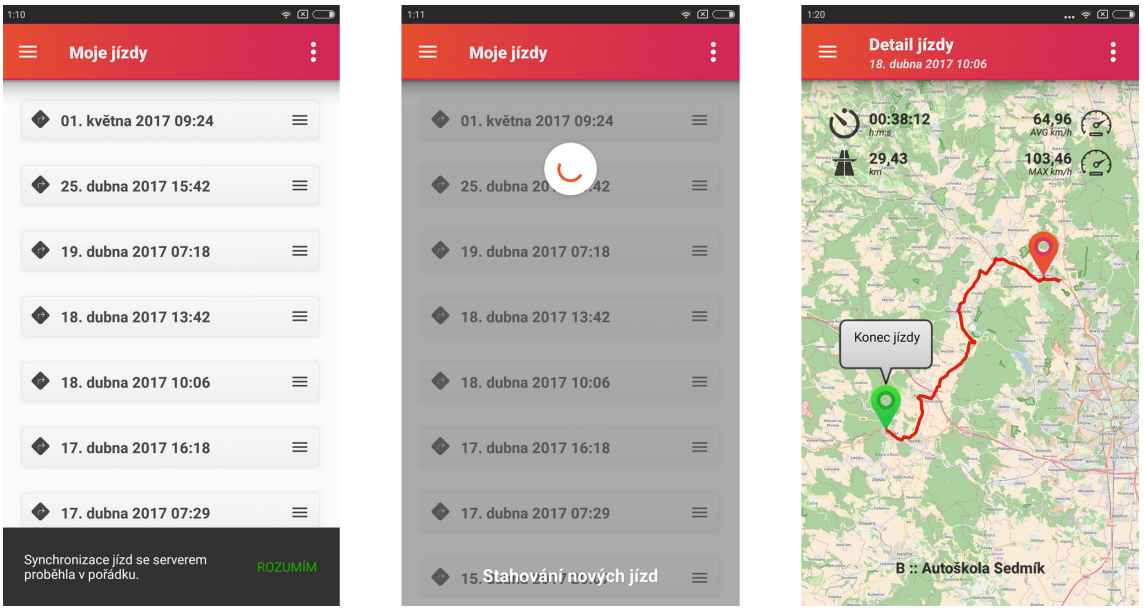
B.4 Mobilní aplikace



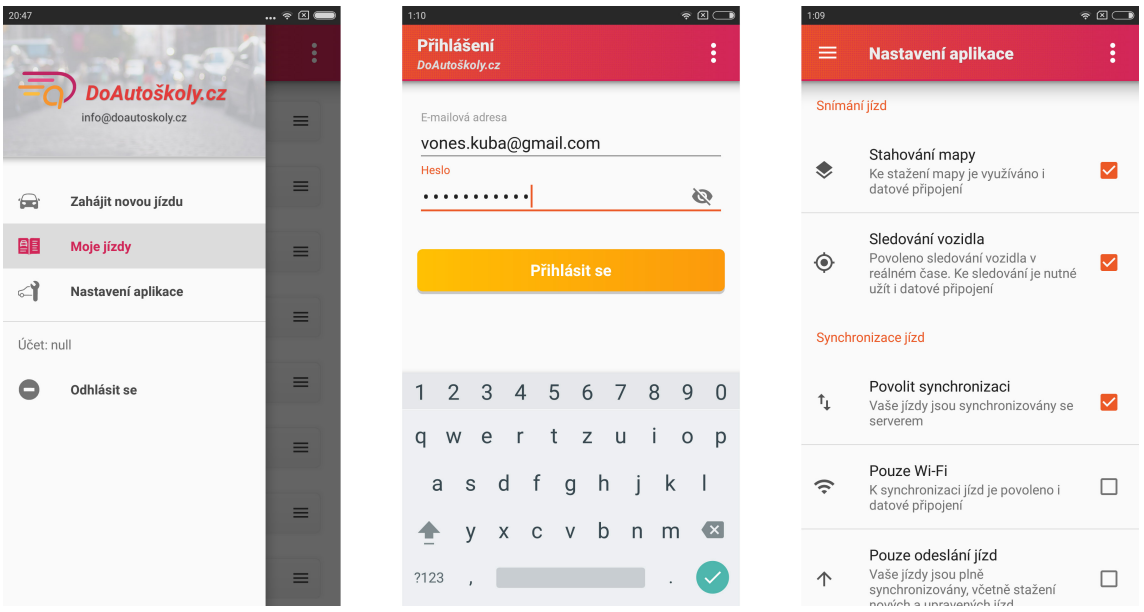
Obrázek B.5: Ukázka mobilní aplikace, která primárně slouží k snímání tras pomocí *GPS*. Konkrétně na tomto obrázku jsou znázorněny tyto fragmenty: zahájení nové jízdy a průběh snímání jízdy (zleva).



Obrázek B.6: Na tomto obrázku je znázorněn fragment průběhu snímání jízdy, který obsahuje po své levé straně dvě ikony reprezentující signál *GPS* a aktivní sledování studenta. Po pravé straně jsou další dvě ikony ukazující aktuální rychlost a ujetou vzdálenost. Jsou zde také vidět ovládací prvky pro ukončení a pozastavení jízdy.



Obrázek B.7: Na tomto obrázku jsou znázorněny tyto fragmenty: seznam ukončených jízd, synchronizace se serverovou aplikací a detail ukončené jízdy (zleva).



Obrázek B.8: Na tomto obrázku jsou znázorněny tyto fragmenty: otevřená navigace s referencemi na další fragmenty, přihlášení k mobilní aplikaci a uživatelské nastavení aplikace (zleva).

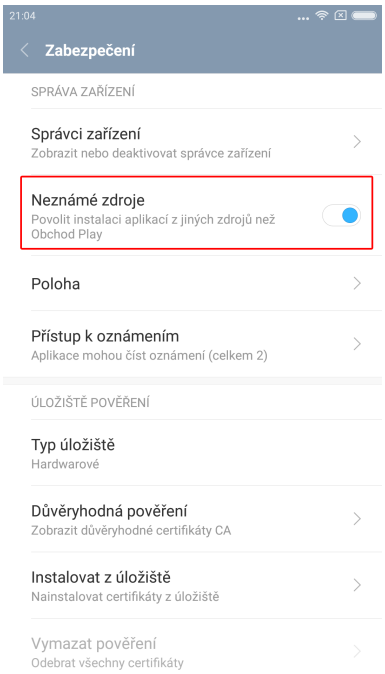
Příloha C

Manuál mobilní aplikace

V této příloze bude nastíněn jednoduchý návod nastiňující netriviální nastavení mobilní aplikace, které je nutné na některých zařízeních učinit. Tyto problémové situace byly nastíněny v kapitole 10.2.3.

C.1 Instalace ze souboru

V reálném použití se aplikace budou jednoduše instalovat pomocí oficiálního *storu*. V Případě instalace ze souboru *.apk* je nutné předem povolit tzv. „*instalaci z neznámých zdrojů*“. Následně je nutné soubor ve formátu *.apk* zkopírovat do paměti telefonu a kliknutím spustit instalaci.



Obrázek C.1: Nastavení instalace z neznámých zdrojů.

C.2 Nastavení aplikace

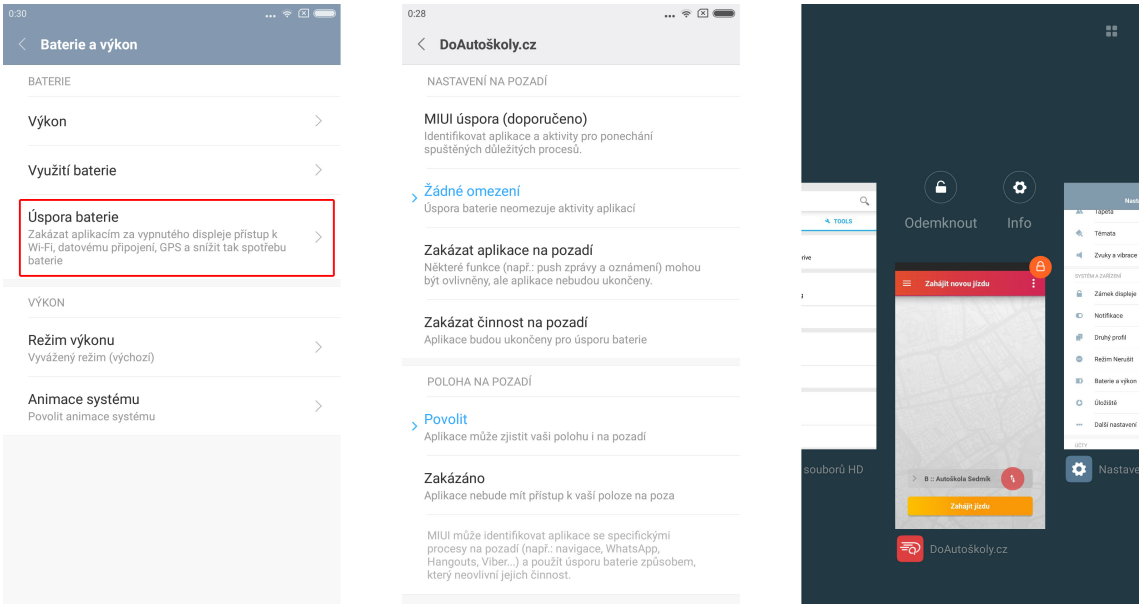
Po nainstalování aplikace a přihlášení uživatele je nutné před spuštěním snímání jízdy zkontrolovat a případně udělit aplikaci patřičná práva:

1. **Zapnuté a povolené *GPS*** – V nastavení telefonu je nutné zapnout a povolit *GPS*. U některých novějších telefonů je nutné *GPS* povolit zvlášť pro každou aplikaci. V případě špatného nastavení se v aplikace zobrazí chybová hláška a nedojde k zapnutí snímání jízdy. Toto nastavení se obvykle nachází:

   - Nastavení -> Nainstalované aplikace -> *DoAutoškoly.cz* -> Správa práv
   - Nastavení -> Povolení -> Správa povolení -> Vaše poloha

2. **Šetření baterie** – V nastavení telefonu je nutné zkontrolovat položku „*baterie a výkon*“. Aplikace po celou dobu měření pracuje na pozadí telefonu a při špatném nastavení by mohlo dojít k přerušení či omezení *GPS*. Bohužel toto nastavení nelze zjistit s předstihem, protože závisí na konkrétním výrobci mobilního zařízení. Aplikace v takovém případě není informována o nové pozici zařízení. Toto nastavení se může lišit právě v závislosti na konkrétním výrobci, pokud jej zařízení vůbec obsahuje, ale obvykle se nachází:

   - Nastavení -> Baterie a výkon -> Úspora baterie -> dále vyberte aplikaci *DoAutoškoly.cz* a nastavte ji „*Žádné omezení*“ a povolit polohu na pozadí.
   - V případě problémů s předchozím bodem lze jednoduše aplikaci po dobu běhu „*uzamknout*“ v seznamu právě spuštěných aplikací.



Obrázek C.2: Demonstrace správného nastavení pro aplikaci *DoAutoškoly.cz*.

Dále je doporučeno zkontrolovat uživatelské nastavení přímo v aplikaci, kde je možné zakázat sledování polohy majitelem autoškoly při snímání jízdy nebo zakázat stahování mapy v závislosti na datovém připojení.